Systems development life cycle sdlc systems

Life

These include questioning why the information system needs to be built, to charting feasibility factors, analyzing issues that are expected to crop up, zeroing in on a certain design and architecture for the system, developing the system, implementing and testing it, and delivering it to the target customer. Each of these activities Is carried out In an orderly manner In a set of well defined developmental phases. This Is why OSDL Is referred to as a process of gradual refinement. That is, each phase takes over where the previous phase left off, and refines what was done before.

The most common development phases in OSDL are: Planning In this phase, " Why does this system need to be built? " is asked and answered appropriately. The system's requirements are clearly defined and a thorough feasibility study Is done from various perspectives Including technical. Financial, and organizational. Analysis This phase Involves problem Identification and solution analysis. Efforts are made to predict the problems that the developers might face while creating the system.

The end-deliverables of this phase determines the method of system creation and provide guidance to the developer. Design Analysis results In decisions regarding system design, which determines the system operation In terms of the process, data, hardware requirements, network infrastructure needs, user interface, etc. Implementation This Is the phase when the actual development happens. The system Is built, tested for defects that are rectified, and finally Installed. Bundled along are other essential more specific phases such as Development and Deployment and Maintenance.

However the above four phases are the most common and accepted ones. From the base structure of OSDL, numerous methodologies have been created to customize the process based on requirements and to improve upon existing methods. Although each of these methodologies follows its own techniques and tepees, they all still fall under the broad spectrum of the 4 phases of the OSDL methodology. There are many different software development methodologies in place today, including the waterfall method, the fountain, the spiral, and agile development and incremental methodologies.

All of them are extensions of three main strategies - Structured Design, Rapid Application Development, and Object Oriented Analysis and Design. Structured Design A structured design method involves a step-by-step approach to a system building process, logically moving from one developmental phase to the next. The deliverable of each phase is first submitted to the customer for approval before starting the next phase. Even within structured design, two distinct approaches exist: a process centered approach aims to complete the work entirely from the perspective of the processes that exist within the system, I. . , it results in a system that is based on process oriented components, while a data based approach shifts the focus to the data that is consumed by and involved in the system. The two most popular structured design methodologies are the Waterfall model and the Parallel Development model. The Waterfall model is a ten step methodology that involves the development of software in a sequential manner, moving from one phase to another in accordance to Scud's fundamental phases.

The Parallel Development model differs from the Waterfall model in that it does not wait for completion of one phase for the next to commence. The entire project is segregated into a set of small projects, and each phase is implemented separately for each project. Therefore, the development process happens concurrently for all the sub-projects. The advantage of any structured methodology is the rigorous and disciplined manner in which it forces velveteen to take place. System requirements are analyzed and understood prior to the actual implementation phase.

However, this same rigor and inflexibility makes methodologies based on structured design completely vulnerable to business changes during the course of development, as it is extremely difficult to go backwards from a certain point in any phase. It might require a complete process restart. Also, it takes a long time to present users with any deliverables, as no part of the system is completely done until the end of the implementation phase. Rapid Application Development methodologies aim to adjust the OSDL phases in such a way that the core functioning part of the system is developed as soon as possible and delivered.

Many RADAR methodologies also try to be flexible to the changes in the business process, by carrying out all the developmental phases almost concurrently. Examples include Prototyping RADAR and Agile Development. RADAR methodologies also signaled the advent of advanced development tools such a code generators and G programming languages such as Microsoft's Visual Basic and Borderland's Delphi, which also contributed significantly to speeding up the development process. RADAR methodologies can be segregated into three types: Phased Development Methodologies

based on Phased Development break system requirements into a set of versions.

Each version is built sequentially and logically. The most fundamental functions go into the first version and the next version is built once the preceding version is completed. Finally, all the versions developed are tailored together to form a complete system. Such methodologies are adept at presenting the crucial part of the system quickly to the end users. Prototyping Prototyping methodologies are best suited for environments where the business process is likely to change during development or a clear idea of the system to be built is not yet available.

The OSDL phases of Analysis, Design, and Implementation are performed concurrently on each cycle, resulting in a prototype, which the project sponsor reviews. Based on the sponsor's feedback, the cycle repeats continually to create more prototypes. The final prototype that meets all requirements becomes the system. Methodologies based on prototyping ensure quick delivery, but do not ensure complete fulfillment of requirements. Throw-away Prototyping Methodologies based on Throw-away Prototyping also develop prototypes, but with a difference. These prototypes are only meant to be the system's visual markers, and deliver no functionality.

The prototype is continually updated based on user comments till it can successfully help visualize the complete system. Based on this dummy or mock-up prototype, the actual system is built. Instead of delivering incomplete systems rapidly within the project time-line, these methodologies

deliver the completed system in a shorter time as compared to other methodologies. Object-oriented Analysis and Design The purpose of creating the object-oriented methodology was to address the lack of compatibility between process and data centered approaches in the OSDL methodology.

It is difficult for a system developed in a process-centric manner to adapt to changes in data, and vice versa. Object Oriented or 00 methodologies and data. Methods that are mapped to such objects carry out activities and processes, and an object also has states referred to as attributes. In such a setup, developers end up focusing on the object, which is the entity that handles processes ND carries data. It avoids the pitfalls of a data-centric or process-centric approach. 00-based approaches have led to the development of multiple object-oriented programming languages, the most well-known being C++ and Java.

An 00 approach in system development: Employs use-cases as the primary modeling tool to chart out the behavior of the system. A use-case describes the interaction between a user and a particular aspect of the system for a certain activity. As they always focus on a single activity and the interaction involved, use-cases are inherently simple. Is architecture-centric, I. , it provides a high level view of the system that is being developed. The chosen architecture dictates the specifications, construction, and documentation of the system.

There are three views that must be supported by the system architecture: Functional View - Use-case diagrams depict the system's behavior from the user's perspective. Static View - ClassResponsibilityCollaboration Cards (CRY)

and class and object diagrams describe the system's structure in terms of the classes, methods, attributes, and relationships of objects within the system. Dynamic View - Unified Modeling Language (ML) tools create sequence diagrams, collaboration diagrams, and object state charts to describe the internal system behavior in terms ofcommunicationbetween objects and changes of state.

Is iterative and incremental. That is, at each iteration of the system development, the system should be closer to meeting all of the requirements. Since OSDL is a process of gradual refinement, the ML diagrams used in 00-based development constantly evolve during the developmental phases. They go from a conceptual, abstract entity in the initial phases to a more complex and detailed entity as the implementation phase nears.