**4. SYSTEM ANALYSiS**

The **Systems Development Life Cycle (SDLC)**, or *Software Development Life Cycle* in [systems engineering](http://en.wikipedia.org/wiki/Systems_engineering), [information systems](http://en.wikipedia.org/wiki/Information_systems) and [software engineering](http://en.wikipedia.org/wiki/Software_engineering), is the process of creating or altering systems, and the models and [methodologies](http://en.wikipedia.org/wiki/Methodologies) that people use to develop these systems.

In software engineering the SDLC concept underpins many kinds of [software development methodologies](http://en.wikipedia.org/wiki/Software_development_methodologies). These methodologies form the framework for planning and controlling the creation of an information system the [software development process](http://en.wikipedia.org/wiki/Software_development_process).

**SOFTWARE MODEL OR ARCHITECTURE ANALYSIS:**

Structured project management techniques (such as an SDLC) enhance management’s control over projects by dividing complex tasks into manageable sections. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. But none of the SDLC models discuss the key issues like Change management, Incident management and Release management processes within the SDLC process, but, it is addressed in the overall project management. In the proposed hypothetical model, the concept of user-developer interaction in the conventional SDLC model has been converted into a three dimensional model which comprises of the user, owner and the developer. In the proposed hypothetical model, the concept of user-developer interaction in the conventional SDLC model has been converted into a three dimensional model which comprises of the user, owner and the developer. The ―one size fits all‖ approach to applying SDLC methodologies is no longer appropriate. We have made an attempt to address the above mentioned defects by using a new hypothetical model for SDLC described elsewhere. The drawback of addressing these management processes under the overall project management is missing of key technical issues pertaining to software development process that is, these issues are talked in the project management at the surface level but not at the ground level.

**WHAT IS SDLC?**

A software cycle deals with various parts and phases from planning to testing and deploying software. All these activities are carried out in different ways, as per the needs. Each way is known as a Software Development Lifecycle Model (SDLC). A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models.

**SDLC models** \* **The Linear model (Waterfall)** - Separate and distinct phases of specification and development. - All activities in linear fashion. - Next phase starts only when first one is complete. \* **Evolutionary development** - Specification and development are interleaved (Spiral, incremental, prototype based, Rapid Application development). - Incremental Model (Waterfall in iteration), **-** RAD(Rapid Application Development) **-** Focus is on developing quality product in less time, - **Spiral Model** - We start from smaller module and keeps on building it like a spiral. It is also called Component based development. \* **Formal systems development** - A mathematical system model is formally transformed to an implementation. \* **Agile Methods.** - Inducing flexibility into development. \* **Reuse-based development** - The system is assembled from existing components.

**The General Model**

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. There are tons of models, and many companies adopt their own, but all have very similar patterns. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced during implementation that is driven by the design. Testing verifies the deliverable of the implementation phase against requirements.

**SDLC Methodology:**

**Spiral Model**

The spiral model is similar to the incremental model, with more emphases placed on risk analysis.  The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.  A\ software project repeatedly passes through these phases in iterations (called Spirals in this model).  The baseline spiral, starting in the planning phase, requirements is gathered and risk is assessed.  Each subsequent spirals builds on the baseline spiral. Requirements are gathered during the planning phase.  In the risk analysis phase, a process is undertaken to identify risk and alternate solutions.  A prototype is produced at the end of the  
risk analysis phase. Software is produced in the engineering phase, along with testing at  
the end of the phase.  The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral. In the spiral model, the angular component represents progress, and the radius of the spiral represents cost. Spiral Life Cycle Model.

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

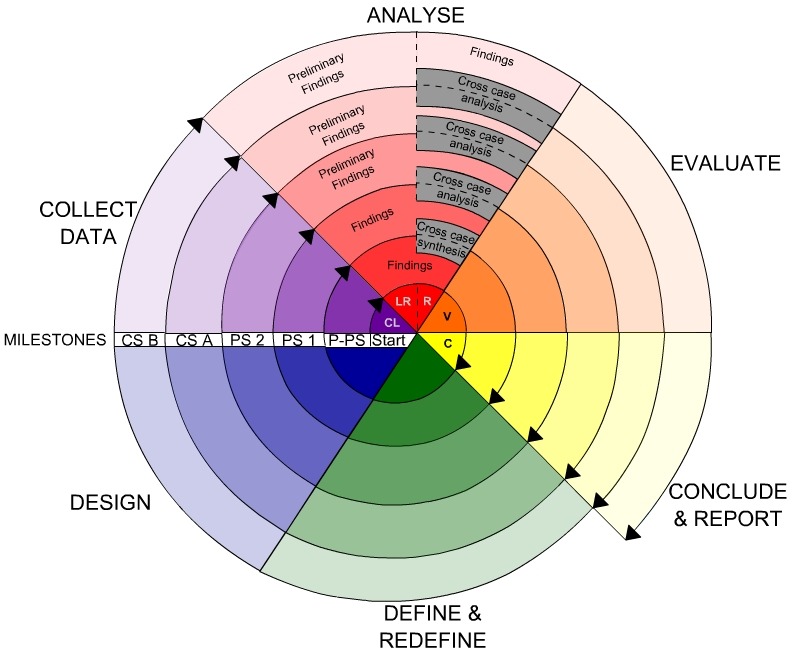
As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning an designing the second prototype.
4. Constructing and testing the second prototype.

* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.



**Fig -Spiral Model**

**Advantages**

* High amount of risk analysis
* Good for large and mission-critical projects.
* Software is produced early in the software life cycle.

**Existing system:**

Almost all the state-of-the-art methods, however, require human experts at certain levels. Most existing methods rely on complex, trained linguistic analyzers (e.g., dependency parsers) to locate phrase mentions, and thus may have unsatisfactory performance on text corpora of new domains and genres without extra but expensive adaption. Our latest domain-independent method SegPhrase outperforms many other approaches but still needs domain experts to first carefully select hundreds of varying quality phrases from millions of candidates, and then annotate them with binary labels. Text indexing algorithms typically filter out stop words and restrict candidate terms to noun phrases. With predefined part-of-speech (POS) rules, one can identify noun phrases as term candidates in POS-tagged documents. Supervised noun phrase chunking techniques, exploit such tagged documents to automatically learn rules for identifying noun phrase boundaries. Other methods may utilize more sophisticated NLP technologies such as dependency parsing to further enhance the precision With candidate terms collected, the next step is to leverage certain statistical measures derived from the corpus to estimate phrase quality. Some methods rely on other reference corpora for the calibration of “termhood”.

**Disadvantages:**

* The dependency on these various kinds of linguistic analyzers, domain-dependent language rules, and expensive human labeling, makes it challenging to extend these approaches to emerging, big, and unrestricted corpora, which may include many different domains, topics, and languages.
* SegPhrase, additional manual labeling effort needed.

**Proposed system:**

A novel automated phrase mining framework, Auto Phrase has two new techniques.1. Robust Positive-Only Distant Training,2. POS-Guided Phrasal Segmentation. AutoPhrase can support any language as long as a general knowledge base in that language is available, the POS-guided phrasal segmentation can be applied in many scenarios. It is worth mentioning that for domain-specific knowledge bases (e.g., MeSH terms in the biomedical domain) and trained POS taggers, the same paradigm applies. In this study, without loss of generality, we only assume the availability of a general knowledge base together with a pre-trained POS tagger.

**Advantages:**

Our main contributions are highlighted as follows:

1. Automated phrase mining

2. a robust positive-only distant training method for phrase quality estimation to minimize the human effort.

3Develop a novel phrasal segmentation model to leverage POS tags to achieve further improvement, when a POS tagger is available.

**Architecture Diagram**

1. View Users Request,
2. Add Candidate,
3. View all Candidate Details,
4. View candidate Access Points,
5. View search Transactions,
6. View Auto Phrase Mining,
7. Positive Pool search,
8. Negative Pool search,
9. View Profile,
10. Security

Accepting all user Information

**Admin**

View user data details

Authorize the Admin

Process all user queries

**Store and retrievals**

**WEB Database**

Registering the User

* 1. **System Requirements Specification**

**5.1 Introduction**

A **Software Requirements Specification** (**SRS**) – a [requirements specification](http://en.wikipedia.org/wiki/Requirements_specification) for a [software system](http://en.wikipedia.org/wiki/Software_system) – is a complete description of the behavior of a system to be developed. It includes a set of [use cases](http://en.wikipedia.org/wiki/Use_case) that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. [Non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) are requirements which impose constraints on the design or implementation (such as [performance engineering](http://en.wikipedia.org/wiki/Performance_engineering) requirements, [quality](http://en.wikipedia.org/wiki/Quality_%28business%29) standards, or design constraints).

**System requirements specification:** A structured collection of information that embodies the requirements of a system. A [business analyst](http://en.wikipedia.org/wiki/Business_analyst), sometimes titled [system analyst](http://en.wikipedia.org/wiki/System_analyst), is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Within the [systems development life cycle](http://en.wikipedia.org/wiki/Systems_development_life_cycle) domain, typically performs a liaison function between the business side of an enterprise and the information technology department or external service providers. Projects are subject to three sorts of requirements:

* [**Business requirements**](http://en.wikipedia.org/wiki/Business_requirements) describe in business terms *what* must be delivered or accomplished to provide value.
* **Product requirements** describe properties of a system or product (which could be one of

several ways to accomplish a set of business requirements.)

* **Process requirements** describe activities performed by the developing organization. For instance, process requirements could specify specific methodologies that must be followed, and constraints that the organization must obey.

Product and process requirements are closely linked. Process requirements often specify the activities that will be performed to satisfy a product requirement. For example, a maximum development cost requirement (a process requirement) may be imposed to help achieve a maximum sales price requirement (a product requirement); a requirement that the product be maintainable (a Product requirement) often is addressed by imposing requirements to follow particular development styles

**5.2 Purpose**

An systems engineering, a **requirement** can be a description of *what* a system must do, referred to as a [Functional Requirement](http://en.wikipedia.org/wiki/Functional_requirements). This type of requirement specifies something that the delivered system must be able to do. Another type of requirement specifies something about the system itself, and how well it performs its functions. Such requirements are often called [Non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements), or 'performance requirements' or 'quality of service requirements.' Examples of such requirements include usability, availability, reliability, supportability, testability and maintainability.

A collection of requirements define the characteristics or features of the desired system. A 'good' list of requirements as far as possible avoids saying *how* the system should implement the requirements, leaving such decisions to the system designer. Specifying how the system should be implemented is called "implementation bias" or "solution engineering". However, *implementation constraints* on the solution may validly be expressed by the future owner, for example for required interfaces to external systems; for interoperability with other systems; and for commonality (e.g. of user interfaces) with other owned products.

In software engineering, the same meanings of requirements apply, except that the focus of interest is the software itself.

**4.3 FUNCTIONAL Requirements**

**Admin:**

In this module, server has to login with valid username and password. After login successful he can do some operations such as View Users Request, Add Candidate, View all Candidate Details, View candidate Access Points, View search Transactions, View Auto Phrase Mining, Positive Pool search, Negative Pool search, View Profile, Security.

**User:**

In this module, Users registers before doing some operation. After registration successful he can login by using valid username and password. After login successful he can do some operations such as Search for Candidate, View MySearch History, View MyPhrase Mining, View Profile, and Security.

**4.4 NON Functional Requirements**

The major non-functional Requirements of the system are as follows

**Usability**

The system is designed with completely automated process hence there is no or less user intervention.

**Reliability**

The system is more reliable because of the qualities that are inherited from the chosen platform java. The code built by using java is more reliable.

**Performance**

This system is developing in the high level languages and using the advanced front-end and back-end technologies it will give response to the end user on client system with in very less time.

**Supportability**

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is having JVM, built into the system.

**Implementation**

The system is implemented in web environment using struts framework. The apache tomcat is used as the web server and windows xp professional is used as the platform.

Interface the user interface is based on Struts provides HTML Tag

**SOFTWARE REQUIREMENTS:**

Operating System : Windows XP/2003 or Linux

User Interface : HTML, CSS

Client-side Scripting : JavaScript

Programming Language : Java

Web Applications : JDBC, Servlet, JSP

IDE/Workbench : My Eclipse 8.6

Database : Oracle 10g

Server Deployment : Tomcat 7.X

**Hardware Requirements:**

Processor : Pentium IV

Hard Disk : 500GB

RAM : 2GB or more