

Software
Methodologies
followed by -



Contents

Topic	Page no
Introduction	2
SDLC	3
Analysis	5
Designing	6
Implementation	7
Testing	8
UAT	11
Summary	12

Document Title	Software Methodology
Date	1 st February 2001
Revision	Version 1.5
Created By	Avinash, Deepesh, Ganapati, Pravin
Reviewed By	G. Vasudev Rao

Software Methodology

Introduction

This document describes about the various phases in System Development Life Cycle (SDLC) and the methodology used by S.K.International during the various phases. The aim of the document is to make the client aware of the practices followed by S.K.International to develop a quality product on time every time.

Software projects are more complex than ever before. It's an established fact that coding is not a major part of any project. Project analysis and planning are instrumental in delivering a quality project on time. Only well-architected applications can scale to meet the demand of Internet developments and easily adapt to changing business conditions.

Just as a house can't be built properly without having an architectural blueprint similarly is not possible to develop a quality project without proper planning of the entire process.

We realize that the ingredients contributing to a success of any project are people, tools that they use and the practices they follow.

After collecting the RS from the client, we break up the project into various activities like planning, analysis, designing, testing, implementation, maintenance, change management, installation, user training and **divide them into various phases** like inception, elaboration, construction and transition. Planning is a continuous process.

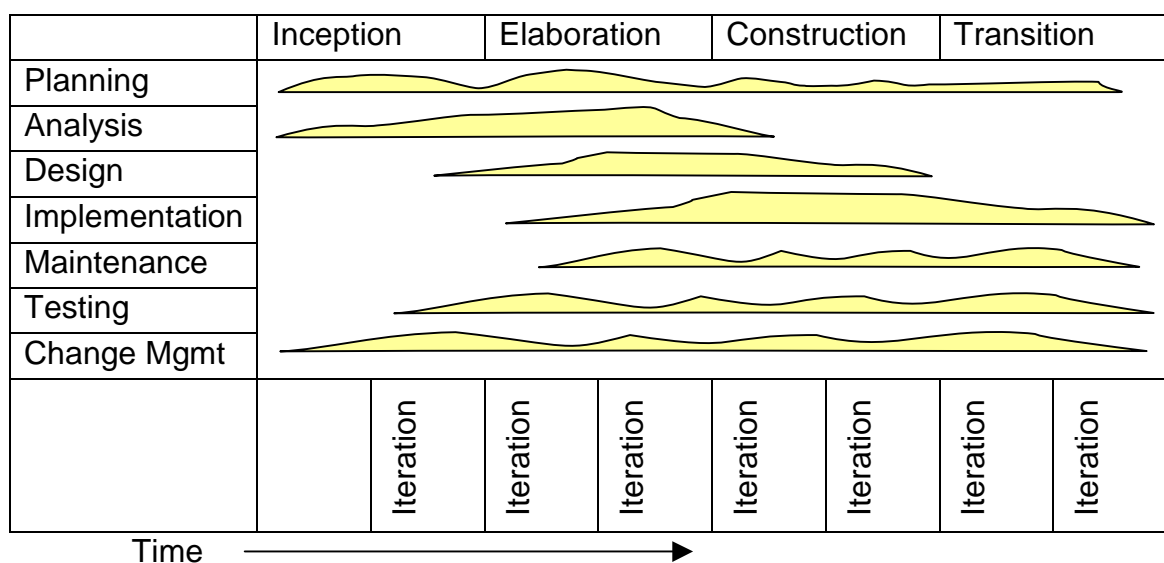
System Development Life Cycle (SDLC)

Phases

A project is broken into four phases

1. **Inception** (Feasibility study) The general project vision is expressed in a preliminary list of requirements, financial estimation taken and a go ahead/no-go ahead decision taken.
2. **Elaboration** starts with requirement analysis based on use cases prepared in inception phase, a complete development plan for the whole project is prepared for the iterations, requirement list for iterations and results of quality assurance. Lot of analysis and design is done in this phase.
3. **Construction** The objective is to develop a software product ready for transition into user community. Most of implementation activities are done in this phase. The output is a beta release.
4. **Transition** This phase consists of transfer of software to its user community. Installation is done and end user training provided. This is done by small group of testers, developers and part-time architect for ensuring overall consistency.

We develop the project **iteratively** as shown below.



- Planning is a continuous activity and includes development plans, measurement of progress and control of project resources.
- Analysis is mainly restricted to elaboration. Here we make user models, project vision and evaluation criteria's.
- Design happens mostly in elaboration phase and we make class diagram, general design of whole application and building of common mechanisms.
- Implementation starts in elaboration phase and mostly occurs in construction phase, with extensive coding, testing etc.
- Maintenance starts with the first version of software definition. Once its placed under version and configuration management (also called versioning).
- Testing happens in most of the phases. Testing proves that software has reached the objectives established and measured by the criteria.
- Change management. In this activity we maintain history of whole project.

Using **Microsoft Project** we chalk out the activities planned in the SDLC. Once this is done we continuously monitor project. This helps in ensuring that the project is running as per schedule and activities are properly implemented.

The workload on each team member is reviewed and if required workload is correspondingly redistributed between developers based on load & expertise levels.

The Actual Process

Analysis

We start with crucial phase of project analysis. **Project Analysis** is foundation on which a software project is built. Our project analysts take care of analyzing the user requirements and he bridges the gap between the client and our software development team.

We have always believed in **OOPS concept** for project analysis as it offers various advantages like we can compare the stability of a model with real life entities. Another advantage is the reusability. For analysis we use visual modeling and we follow **Unified Modeling Language (UML)**. UML is the industries defacto standard when it comes to visual modeling. UML is a common language that can be understood by the machine as well as humans. The UML allows analysts, software architects, and developers to specify, visualize and document an entire system. UML as a **common language** allows many different contributors with diverse perspectives to communicate on common ground. Using a single modeling tool through the development lifecycle helps ensure that you are building the right system. The architecture model can be traced back to both the business process model and the system requirements.

We use **Rational Rose** as a tool supporting UML. Rose allows users to model their components and interfaces more effectively. Rose supports round trip engineering allowing one to move easily from analysis to design to implementation and back to analysis again.

We take the requirements from our client. The requirements then raise the following questions “What to do?” ” How to do?” and “What skills to use?” After a details study and after understanding the requirements we need to outline the functionality of the system. Also we have to understand what will be handled by the system and what will be handled outside the system. Next we first find actors. **Actors** are entities who will be exchanging or interacting with the system. When the first outlines of the actors are complete the next step to look for the systems uses cases.

Use case is pattern of behavior that system exhibits. Each use case is a sequence of related transactions performed by the actor on the system. The best way to find use cases is to consider what each actor requires from the system.

Use cases are very important and must include all the tasks an actor may want the system to perform, including reading or writing data into system. Once we find out use cases we have the idea what system will do, what it won't do. The use case should list of the core responsibilities of the components. These are points that will be used to validate the design.

Once we have analyzed the system by making defining actors and identifying the use cases we now know "What to do".

Then we try to find answer to question "how to do". Class diagrams help in answering this question.

Designing

The **class diagrams** help us to visualize structure and the behavior of the system and helps to know what objects are contained in our system. In general they express static structure of the system. The class diagrams are based on use cases.

We identify classes and draw them using class diagrams giving their attributes and methods (also identifying whether they are public/private). Next step is to find out relationships between classes through associations (bi-directional access), **Interface Inheritance, Aggregation** (part of or uses). We define roles, multiplicity, navigations etc.

After that we have to show interaction between the components with respect to time using **collaboration and/or sequence diagram**. This makes us clear as to how overall the system will behave and what interaction will take place between the components.

Database Designing

Once the class diagrams are over we can then go for data modeling. Rose extends the UML to database design by mapping between object and data models. Thus the gap between a business analyst and a database designer is bridged.

Data Models are similar to class diagrams. Data models are represented like classes but with a symbol on top right corner. For every table we have to define

what fields (columns) it contains and **the database triggers/procedures** written on it. Also we have to identify dependencies between two tables. We define **primary key constraints, foreign key constraints unique constraints etc.**

Thus having data model mapped to classes, object sequences etc to database elements its easy to transform classes into tables or from tables to classes. This helps our entire team to work seamlessly and any requirement changes can easily be tracked and accordingly incorporated.

Rose supports oops languages like Java and can convert classes into code. Thus we can get easy to use templates. Rose can also generate Data Description Language (DDL) for database applications. Rational Rose also supports reverse engineering so that any new methods or attributes added to a class can be reflected in the class diagram thus maintaining consistency.

Component Modeling

We also do Component modeling. The component diagrams illustrate organization and dependencies between software components. It provides modeling contract to visualize physical nature of the system.

Once all our diagrams are in place and we fully understand the working of the system we construct a deployment diagram that gives the idea as to how the system will actually be deployed. **Deployment diagrams** are for configuring runtime processing elements and software process living in them. We denote the devices through nodes. Interconnection of nodes is through association

Implementation

Once class diagrams and data model are through the development activity can start. i.e. the beginning of the implementation phase. We then identify the **dependent and independent modules** and according allocate modules to programmers.

Software development is done in accordance to GUI standards, coding standards and design specifications to assure quality of the product. **GUI standards** are arrived at, to **achieve consistency** in different aspects of user interface. Coding standards are defined and followed for ease of code maintenance and Uniformity throughout the SDLC.

Testing

Once a module is ready it has to pass through various levels of rigorous testing. We firmly believe that **quality** is a distinguishing attribute of a system indicating the **degree of excellence**.

In many software engineering methodologies, the testing phase is a separate phase, which is performed by a different team after the implementation is completed. There is merit in this approach; it is hard to see one's own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read and re-read the material many times. Unfortunately, delegating testing to another team leads to a slack attitude regarding quality by the implementation team.

Alternatively, another approach is to delegate testing to the whole organization. If the teams are to be known as craftsmen, then the teams should be responsible for establishing high quality across all phases. Sometimes, an attitude change must take place to guarantee quality.

The testing technique is from the perspective of the system provider. Because it is nearly impossible to duplicate every possible client's environment and because systems are released with yet-to-be-discovered errors, the client plays an important, though reluctant, role in testing.

We use the following tests for ensuring quality

- **Regression Test**
- **Internal Testing**
- **Unit Testing**
- **Application/Integrated Testing**
- **Stress Testing**

Regression Test

Quality is usually appraised by a collection of regression tests forming a suite of programs that test one or more features of the system.

A regression test is written and the results are generated. If the results generate an error, then the offending bug is corrected. A valid regression test generates verified results. These verified results are called the "gold standard." Ideally, the validity of a test result is driven by the requirement document; in practice, the implementation team is responsible for validity interpretation.

The tests are collected, as well as their gold standard results, into a regression test suite. As development continues, more tests are added, while old tests may remain valid. If this is the case, the old test results are altered in the "gold standard" to match the current expectations. The test suite is run generating new results. These new results are then compared with the gold-standard results. If they differ, then a potential new fault has entered the system. The fault is corrected and the development continues. This mechanism detects when new development invalidates existing development, and thus prevents the system from regressing into a fault state.

There are four major focuses of regression testing used to assure quality. A summary is found in the following Table.

Table 1.0: Categories of Quality

Focus	Note	Team Responsibility
Internal	Make sure all internal, non-client-visible components work well.	Implementation Team
Unit	Make sure all client-visible components work well.	Implementation and Design Teams
Application	Make sure the application can complete all scenarios.	Analysis Team
Stress	Run the application in an environment that is more stressful than the target environment.	All Teams

Internal Testing

Internal testing deals with low-level implementation. Here each function or component is tested. The implementation team accomplishes this testing. Internal limits are tested here.

Unit Testing

Unit testing deals with testing a unit as a whole. This would test the interaction of many functions but confine the test within one unit. The exact scope of a unit is

left to interpretation. Supporting test code, sometimes called scaffolding, may be necessary to support an individual test. This type of testing is driven by the architecture and the implementation teams. This focus is also called black-box testing because only the details of the interface are visible to the test. Limits that are global to a unit are tested here.

In software testing, one particular test may need some supporting software. This software establishes an environment around the test. Only when this environment is established can a correct evaluation of the test take place. The scaffolding software may establish state and values for data structures as well as providing dummy external functions for the test. Different scaffolding software may be needed from one test to another test. Scaffolding software rarely is considered part of the system.

Application/Integration Testing

Application testing deals with tests for the entire application. This is driven by the scenarios from the analysis team. Application limits and features are tested here. The application must successfully execute all scenarios before it is ready for general client availability. After all, the scenarios are a part of the requirement document and measure success. Application testing represents the bulk of the testing done by industry.

Unlike the internal and unit testing, which are programmed, these test are usually driven by scripts that run the system with a collection of parameters and collect results. In the past, these scripts may have been written by hand but in many modern systems this process can be automated.

Stress/Load Testing

Stress testing deals with the **quality of the application** in the environment. The idea is to create an environment more demanding of the application than the application would experience under normal workloads. This is the hardest and most complex category of testing to accomplish and it requires a joint effort from all teams.

A test environment is established with many testing stations. At each station, a script is exercising the system. These scripts are usually based on the regression suite. More and more stations are added, all simultaneous hammering on the system, until the system breaks. The system is repaired and the stress test is repeated until a level of stress is reached that is higher than expected to be present at a client site.

Race conditions and memory leaks are often found under stress testing. A race condition is a conflict between at least two tests. Each test works correctly when done in isolation. When the two tests are run in parallel, one or both of the tests fail. This is usually due to an incorrectly managed lock.

A memory leak happens when a test leaves allocated memory behind and does not correctly return the memory to the memory allocation scheme. The test seems to run correctly, but after being exercised several times, available memory is reduced until the system fails.

UAT

The purpose of acceptance testing is to demonstrate that the **system meets its requirements in the operational environment.**

User acceptance testing is normally the final stage of testing performed on a system. It is usually planned as a final confidence-building step for the users.

UAT has the following advantages.

- Users are independent from the developers of the system so are more objective
- Users understand the business requirements, so can prepare tests and test data which are realistic
- Users define the context in which the system will be used so can better assess it's fitness for purpose
- Users have a vested interest in ensuring that the system is of high quality so are motivated to perform rigorous tests.

We prepare a complete test plan and correct deficiencies found during testing We refine the documentation based on inputs from the test team and thus assure that the completed system is delivered.

Component Development Model Testing

If EJB's has been used in the project its crucial to make sure that your EJBs are scalable, before you go live. In this case we use BeanTest a tool from Empirix. We check the scalability of the EJB via web application Extensive reporting tools provided by BeanTest are capable of providing details about **the response time, response time per method**, and exceptions throws as the number of users increase.

After testing the system has to surpass the load test.

Any web-based application is developed keeping in mind that unlimited users may use the system at any given point of time. As the number of user increase with time, the load on the application server increases. Also with the rapid development in the web technologies, the stability of the application becomes very crucial. As more and more corporates adopt web technology to create business-critical commerce and enterprise applications, the need for fast and accurate application testing becomes essential.

Considering the above factors we use the tools provided by **eTest Suite from RSW** .It is one of the most easy-to-use, seamlessly integrated testing solution optimized for web applications. It provides everything that a business-critical web application should support to ensure scalability and reliability.

Testing tools

We use **eTester for Functional/Regression testing**. We keep track of all the objects that are present on every page and automatically insert test cases to validate them. Each component on each page is represented graphically in Visual Script to give a better idea of the actions performed by the objects.

We use **eLoad** an accurate tool **to test the scalability** of our client's e-Business applications. We simulate thousands of virtual users accessing the applications simultaneously. This helps significantly to know the effect of load on the application and thus test its performance. The reports thus generated will give a clear idea to decide on the system architecture, tuning and hosting alternatives.

We use **eMonitor** to closely check for the **smooth functioning of the entire application for 24/7**.

Another tool we use **is eReporter**. We get a crystal clear picture of the complete testing process, accurately pointing out the flaws/bottle-necks in the application. By correlating user response times with performance statistics of the system under test, e-Reporter identifies where performance bottlenecks exist and provides essential data for capacity planning.

Summary

We at S.K.International are committed to following the **Developer to Developer concept (D2D)**. D2D covers everything from designing of workflow to collaboration between the analyst, designers, developers, testers etc. We implement D2D by following various practices like

- Proper planning of modules
- Giving specific roles and responsibilities to team members depending upon workload, experience and skill sets.
- Constant review of the project progress using tools like MS Project
- Periodic meeting and future planning and addressing various issues regarding the project enhancements.
- Hierarchical structure of organization gives more accountability.

The following people participate in the project

- Project Manger-is the main person who drives everything
- Project Leader-puts the entire team together, manages people and co-ordinates all the activities
- Analyst-understands all the requirements of the client
- Designer prepares models and does data base designing
- Developers-these are people who do all the coding work
- System Administrator-manages the hardware infrastructure
- Integrator-brings together all the different modules
- Library Manager-ensures the archiving and saving of all development artifacts
- Tester-ensures preparation and execution of test cases.
- Documentation writer-prepares user manual and overall documentations

The project is divided into various phases.

- **Inception:** In this phase we study the operational, technical and financial feasibility
- **Elaboration:** In this phase a complete development plan for the whole project. Plan for the iterations is prepared. Lot of analysis and design is done in this phase.
- **Construction** The objective is to develop a software product ready for transition into user community. Most of implementation activities are done in this phase. A beta release is ready at the end of this phase

- **Transition** This phase consists of transfer of software to its user community. Installation is done and end user training provided

Each phase consists of various activities like

- **Analysis:** A through understanding of user requirements and preparation of user case diagrams based on UML.
- **Designing:** Preparation of general design of entire application and case diagrams using UML
- **Database Designing:** Consists of designing the tables, writing triggers and stored procedures.
- **Implementation/Development:** This activity extensive coding is done. Business Logic implemented.
- **Testing:** This activity involves preparation of varied range of test cases with live, ambiguous test data to ensure robustness and reliability. We perform tests like unit test, integration test, regression test, stress test, internal test
- **Maintenance:** This activity includes modifications made to the software once it's placed under version and configuration management.
- **UAT:** This activity comprises of Installation of application at the clients end and imparting end user training.

Following is the list of tools we use in SDLC

- **UML:** We using UML for Visual Modeling and prepare a blue print for the entire project.
- **Rational Rose:** We use this tool to implement UML. The entire analysis done using Rose. We use Rose to prepare to prepare use cases and various diagrams like class diagram, collaboration diagram, sequence diagrams and data model.
- **Rational Soda:** We use this tool for automating the documentation of the entire process.
- **Microsoft MS Project:** This tool is used for planning and monitoring the various schedules and smooth implementation of project plans.
- **RSW eTester:** We use eTester for Functional/Regression testing
- **RSW eLoad:** We use eLoad an accurate tool to test the scalability of our client's e-Business applications.
- **RSW eMonitor:** We use eMonitor to closely check for the smooth functioning of the entire application for 24/7.
- **RSW eReporter:** We use is eReporter to get a crystal clear picture of the complete testing process, accurately pointing out the flaws/bottle-necks in the application.

- **Empirix BeanTest:** We check the scalability of the EJB via web application using BeanTest .We check response time, response time per method, and exceptions throws as the number of users increase.

Thus by following the efficient software engineering methodology our clients are benefited by the excellent performance they get out of the application, not to mention the fact that they get the project on time, every time.