Volume 2, No. 3, May-June 2011

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Software Development Life Cycle: A Detailed Study

Ms Namrata Jain*
Asst. Professor
Jain Arts, Science & Commerce College
Mandsaur (MP), India
E-mail:namrata_12664@rediff.com

Anurag Jain
Jr. Software Developer
SB InfoTech
Indore (MP), India
E-mail:anurag_51787@rediff.com

Abstract: A software development process, also known as a software development life cycle (SDLC), is a structure imposed on the development of a software product. Similar terms include software life cycle and software process. It is often considered a subset of systems development life cycle. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Some people consider a lifecycle model a more general term and a software development process a more specific term. For example, there are many specific software development processes that 'fit' the spiral lifecycle model. ISO 12207 is an ISO standard for software lifecycle processes.

It aims to be the standard that defines all the tasks required for developing and maintaining software. Software Development Life Cycle or SDLC is a model of a detailed plan on how to create, develop, implement and eventually fold the software. It's a complete plan outlining how the software will be born, raised and eventually be retired from its function. The Software development life cycle (SDLC) is the entire process of formal, logical steps taken to develop a software product.

Keywords: SDLC Models, Software Planning, Requirements , Analysis, Design and Testing.

I. INTRODUCTION

A software development process is a structure imposed on the development of a software product. Synonyms include software life cycle and software process. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process [5 and 16]. **Software Development Lifecycle** (SDLC) provides structure so that team members and project stakeholders all understand the current state of the project. It supports visibility and predictability while enabling project teams to make specifics choices that achieve the business goals and constraints. Construx can provide guidance, support, and expertise as you define your SDLC [15][13].

A common misconception is that methodology is synonymous with technology or industry constructs. Structured programming, Event Driven, Client-Server, Object-Oriented, n- Tier, and Service-Oriented Development of Applications (SODA) are all examples of constructs that have evolved over time to deliver faster or better systems. They are not methodologies. A methodology is the process within which any or all of these might be utilized and controlled to deliver the solutions. While technology constructs may influence your selection of a methodology in the short term, value to the business should be the predominant driver of methodology selection. Additionally, the type of organization [13][15][12]
and its focus may significantly influence the selection. Case in point; an Information Technology (IT) or Management Information Systems (MIS) organization will likely select a significantly different methodology from that of a Line of Business (LOB) or Functional organization.

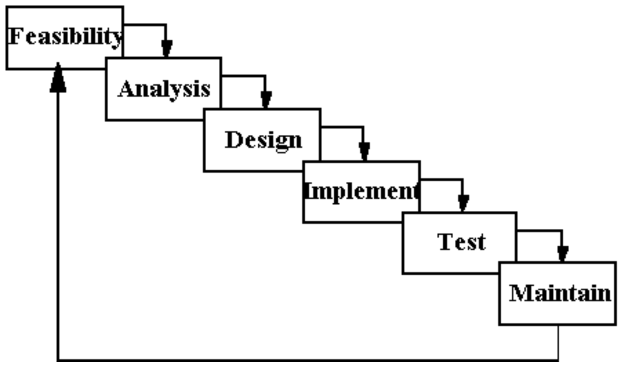
The large and growing body of software development organizations implement process methodologies. Many of them are in the defense industry, which in the U.S. requires a rating based on 'process models' to obtain contracts.
A decades-long goal has been to find repeatable, predictable processes that improve productivity and quality. Some try to systematize or formalize the seemingly unruly task of writing software[11][5]. Others apply project management techniques to writing software. Without project management, software projects can easily be delivered late or over budget. With large numbers of software projects not meeting their expectations in terms of functionality, cost, or delivery schedule, effective project management appears to be lacking.
Organizations may create a Software Engineering Process Group (SEPG), which is the focal point for process improvement. Composed of line practitioners who have varied skills, the group is at the center of the collaborative effort of everyone in the organization who is involved with software engineering process improvement [4][2][13].
In a rapidly changing world people' s needs are also changing rapidly. From simple additions to their car engines to a new technology that should be launched online people will always lack something. It is for this reason that you will always find experts from different fields working on new ideas everyday.

II. SDLC MODEL

A framework that describes the activities performed at each stage of a software development project [11].

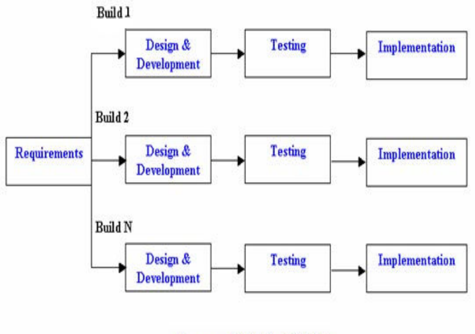
A. Waterfall Model

• Requirements – defines needed information, function, behavior, performance and interfaces.
• Design – data structures, software architecture, interface representations, algorithmic details.
• Implementation – source code, database, user documentation, testing [11][3][7][6].



B. Incremental Model

The Incremental methodology is a derivative of the Waterfall. It maintains a series of phases
which are distinct and cascading in nature. Each phase is dependent on the preceding phase
before it can begin and requires a defined set of inputs from the prior phase [14][6]. However, as the
graphic below portrays, in the design phase development is broken into a series of increments
that can be constructed sequentially or in parallel. The methodology then continues focusing
only on achieving the subset of requirements for that development increment. The process
continues all the way through Implementation. Increments can be discrete components (e.g.,
database build), functionality (e.g., order entry), or integration activities (e.g., integrating a
Human Resources package with your Enterprise Resource Planning application). Again, subsequent
phases do not change the requirements but rather build upon them in driving to completion [15][13][16][10].



Incremental Life Cycle Model

C. Evolutionary (also Known as Iterative)

The Evolutionary methodology also maintains a series of phases that are distinct and cascading in nature. As in the other methodologies, each phase is dependent on the preceding phase before it can begin and requires a defined set of inputs from the prior phase [14][16]. As the graphic below portrays, the Evolutionary methodology is similar to the Incremental in that during the design phase development is broken into a distinct increment or subset of requirements. However, only this limited set of requirements is constructed through to implementation. The process then repeats itself with the remaining requirements becoming an input to a new requirements phase. The "left over" requirements are give consideration for development along with any new functionality or changes. Another Iteration of the process is accomplished through implementation with the result being an "Evolved" form of the same software product. This cycle continues with the full functionality "Evolving" overtime as multiple iterations are completed [9][13][15].

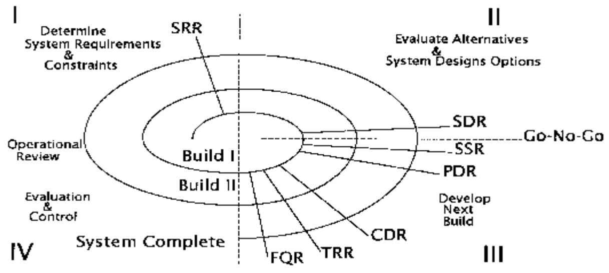
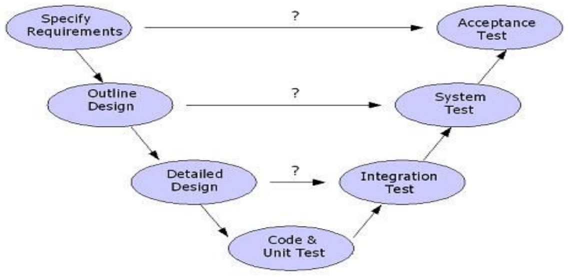


Figure 3: Evolutionary Model

V-SHAPED SDLC MODEL

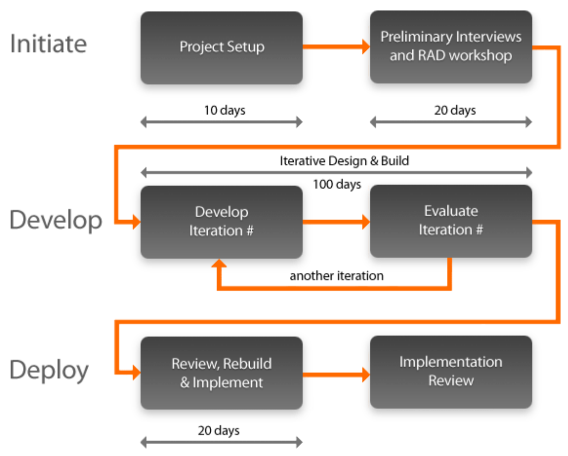
• A variant of the Waterfall that emphasizes the verification and validation of the product.
• Testing of the product is planned in parallel with a corresponding phase of development [11][13][16][8].



D. Rapid Application Model (RAD)

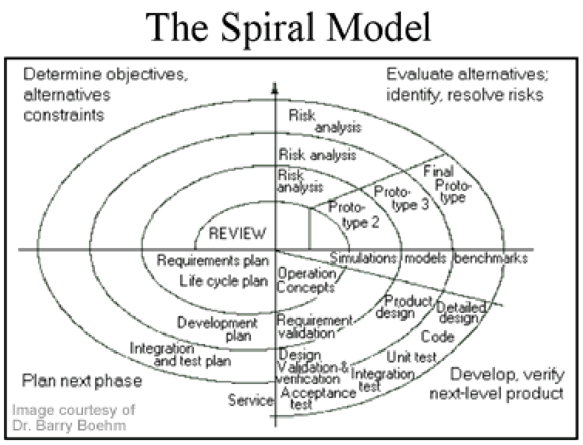
The RAD methodology is a significant departure from the other methodologies in that it is time driven rather than requirements driven. Surely requirements are what define the functionality of the software [16][13][15].

• Requirements planning phase (a workshop utilizing structured discussion of business problems)
• User description phase – automated tools capture information from users
• Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. ("Do until done")
• Cutover phase -- installation of the system, user acceptance testing and user training [15][14][2].



9. Spiral Model

Much as the other methodologies, the Spiral methodology maintains a series of phases
that are distinct and cascading in nature [3]. As in the other methodologies, each phase is
dependent on the preceding phase before it can begin and requires a defined set of inputs
from the prior phase. However, as the graphic below portrays, the Spiral methodology
iterates within the Requirements and Design phases. Unlike the other models, multiple
iterations are utilized to better define requirements and design by assessing risk, simulating, and validating progress. The Spiral methodology also relies heavily on the use and evolution of prototypes to help define requirements and design [13][14][15][16][6].



Investigation
– Identify problems or opportunities
• **Systems Analysis**
– How can we solve the problem
• **Systems Design**
– Select and plan the best solution
• **Systems Implementation**
– Place solution into effect
• **Systems Maintenance and Review**
- Evaluate the results of the solution [15][11].

IV. SOFTWARE REQUIREMENTS

All requirements captured will have clearly stated source (customer contact name, title, and phone number), description, business case, and detailed specifications (if any) [15][2]

- Requirements gathered from existing customers at minimum twice a year and when opportunity presents (source Product Managers)
- Market drivers defined by competitive analysis (source Marketing)
- Requirements from bugs found in existing product releases (source Operations)
- Other requirements gathered (source developers and company architect) [13][6][12].

V. SOFTWARE ANALYSIS

The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished. This phase defines the problem that the customer is trying to solve. The deliverable result at the end of this phase is a requirement document. Ideally, this document states in a clear and precise fashion what is to be built. This analysis represents the ``what'' phase. The requirement document tries to capture the requirements from the customer's perspective by defining goals and interactions at a level removed from the implementation details. The requirement document may be expressed in a formal language based on mathematical logic. Traditionally, the requirement document is written in English or another written language [2][16][1][12].
The requirement document does not specify the architectural or implementation details, but specifies information at the higher level of description. The problem statement, the customer's expectations, and the criteria for success are examples of high-level descriptions. There is a fuzzy line between high-level descriptions and low-level details.
Sometimes, if an exact engineering detail needs to be specified, this detail will also appear in the requirement document. This is the exception and should not be the rule. These exceptions occur for many reasons including maintaining the consistency with other established systems, availability of particular options, customer's demands, and to establish, at the requirement level, a particular architecture vision. An example of a low-level detail that might appear in the requirement document is the usage of a particular vendor's product line, or the usage of some accepted computer industry standard, or a constraint on the image size of the application [13,16 and 5].
Top-down and bottom-up approaches force a greater distinction between high levels and low levels of detail.

Interactive approaches lead to the refinement of those details.

The requirement descriptions of the things in the system and their actions does not imply an architecture design rather a description of the artifacts of the system and how they behave, from the customer's perspective. Later, in the design phase, these requirement descriptions are mapped into computer science based primitives, such as lists, stacks, trees, graphs, algorithms, and data structures [15][12].

VI. SOFTWARE DESIGN

Software design is a process of problem solving and planning for a software solution. After the purpose and specifications of software are determined, software developers will design or employ designers to develop a plan for a solution. It includes low-level component and algorithm implementation issues as well as the architectural view [11][3].

Design team will "initially" consist of the following members

- Engineering Manager
- Senior Product Architect
- Development Team Lead

Design document created by developer will be presented to design team and representatives of QA for a detailed review. Developer will be asked to explain the overall goal his/her design, review the changes to the code in detail, and field questions regarding the impact of the changes on the product as well as the compatibility with previous releases [13].

VII. SOFTWARE TESTING

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.[1] Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects) [1][3].

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

1. meets the business and technical requirements that guided its design and development;
2. works as expected; and
3. can be implemented with the same characteristics**.**

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted [5 and 3].

QA Testing phases will begin once the design document, test plan and test matrixes have been approved. QA will then perform the following testing procedures:[15]

- Functional Testing
- Regression Testing
- Performance Testing
- Load Testing
- Manual Testing
- Exploratory Testing

VIII. CONCLUSION

While selecting the right SDLC methodology is challenging, the challenge is not insurmountable. With a clear understanding of the business and a framework for guidance, selecting a fitting SDLC can be readily achieved. A proper methodology in a maturing environment will enable the business to build software that assists the business in realizing its value proposition [3][5].

IX. REFERENCES

[1] http://www.extremeprogramming.org

[2] http://c2.com/cgi/wiki?ExtremeProgrammingRoad map

[3] http://www.google.co.in/

[4] http://www.xprogramming.com/

[5] http://en.wikipedia.org/wiki/Software_development

[6] http://codebetter.com/raymondlewallen/2005/07/13 /software-development-life-cycle-models/

[7] http://infolab.stanford.edu/~burback/watersluice/no de4.html

[8] http://www.clarotesting.com/userimages/vmodel3.J PG

[9] http://condor.depaul.edu/sjost/hci430/documents/pr ototypes general/Dolan_files/xt94g01i.gif

[10] http://www.qualitytesting.info/profiles/blogs/sdlc-incremental-model

[11] http://cab.org.in/Lists/Knowledge%20Bank/Attach ments/83/SDLC.pdf

[12] www.stylusinc.com/Common/.../SoftwarePhilosop hy.php

[13] Pressman - Software Engineering: A Practitioner's Approach, 6th edition, SPANISH

[14] Selecting a Software Development Life Cycle

[15] (SDLC) Methodology ,A Practical Decision Framework to Maximize Business Value. By : Dr. Alan E. Dillman

[16] Component-Based Software Engineering. By: Thomas Jell