International Journal of Computer Science and Software Engineering (IJCSSE), Volume 5, Issue 1, January 2016
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 5-10

Spiral Increment Reuse (SIR) Software Model

A. SANJAY KUMAR[1], B. Dr. RAHUL RISHI[2] and C. Dr. RAJKUMAR[3]

[1] University Campus School, MDU, Rohtak, India

[2, 3] University Institute of Engineering and Technology, MDU. Rohtak, India

[1]*sheoranincampus@gmail.com,* [2]*rahulrishi@rediffmail.com*

ABSTRACT

The success or failure of any project is very largely depends on its SDLC (Software Development Life Cycle) model. In this paper a new software development lifecycle model is designed using incremental spiral lifecycle model with software reuse concept. The proposed model is designed in such a way that it combines the capability of spiral and incremental model with reusability. The software development process starts with the requirement analysis and identification of required components with the help of SRS followed by component design. Required component are identified from RCL (Reuse Component Library). The components are modified according the requirement and if components are not available, they are rapidly developed and after testing it is stored in TCR (Tested Component Repository). This SDLC model will increase the quality and productivity of project, delivered in shorter tenure and in lower cost.

Keywords: *SDLC Model, Development Phases, Requirements, Risk analysis, Quality and Productivity.*

1. INTRODUCTION

In early days of computing, software developers develop the software in their own way, using their own methods. In the 1970 Royce proposed the waterfall model, which becomes very popular and widely used. Software developers understand the value of SDLC in the development of software projects. A life cycle or software process model prescribes the different activities that need to be carried out to develop a software product and the sequencing of these activities. Later Iterative and Incremental SDLC is developed in response to the weaknesses of the waterfall model. Incremental development is a scheduling and staging strategy in which the various parts of the system are developed at different times or rates and integrated as they are completed. To increase the quality and productivity of software development process, it is essential to choose the right SDLC methodology for your project. Choosing the wrong software methodology will add time to the development cycle, which increases the budget and likely to prevent the delivery of the project on time. It is well known that reuse components have less error and more reliability. So, if the SDLC is accompanied by software reuse or by applying reuse component will increase the quality and productivity of software project and will be delivered in shorter time in low budget. 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.[1]. A prescriptive model prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order.

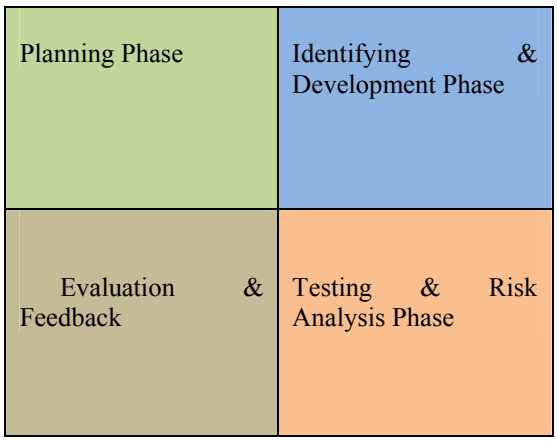

Fig.1. (a)Four Phases of SDLC.

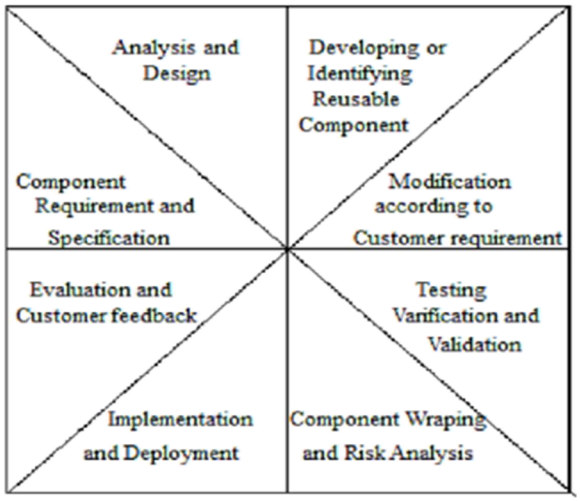

Fig. 1. (b) Eight Phases of SDLC

Every SDLC divides the software development process into number of phases. These are planning, requirement, analysis etc. as shown in fig.1.

2. SOFTWARE DEVELOPMENT MODELS

2.1 Incremental Software

Incremental SDLC is developed in response to the weaknesses of the waterfall model. It is a component based software (CBS) development model. Incremental development is a scheduling and staging strategy in which the various parts of the system are developed at different times or rates and integrated as they are completed [2]. Incremental development life cycle model supports the reusability of software components.

2.2 Spiral model

The Spiral model was proposed by Barry Boehm in 1988. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration matters. Spiral model is an evolutionary software process model that couples the iterative nature of prototype with the controlled and systematic aspect of waterfall model.

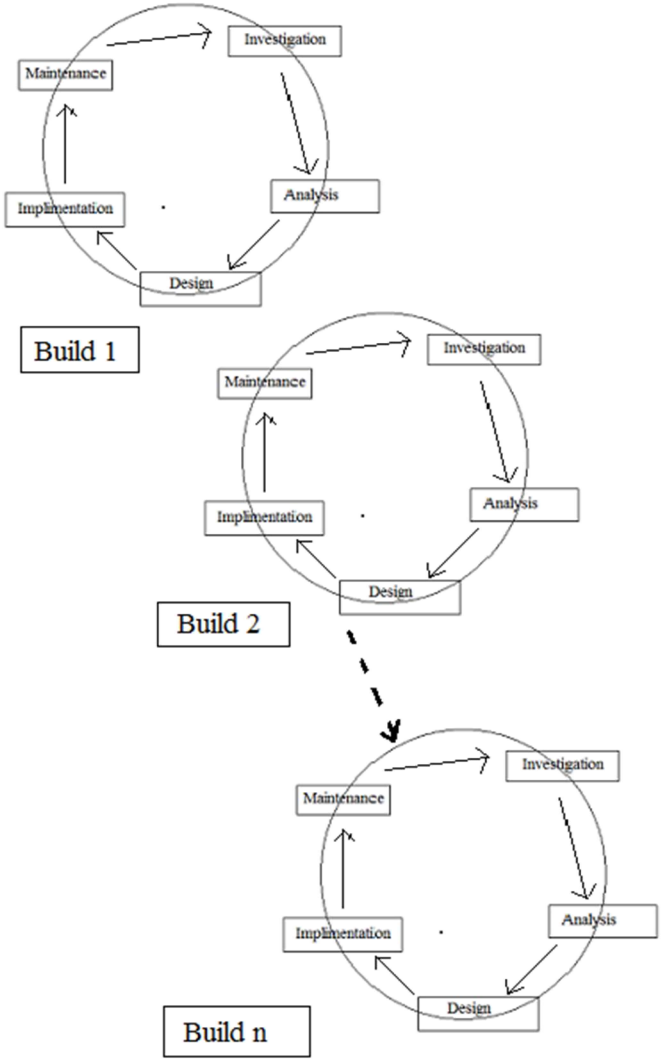

Fig. 2. Incremental Development Software

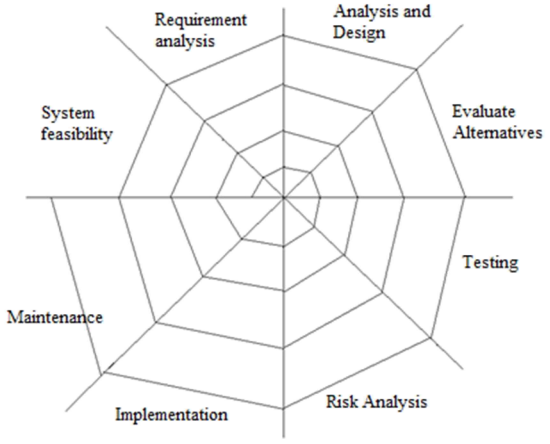

Fig. 3. Barry Boehm Spiral Model

A spiral model is made up of different set of framework activities made by the software engineering team. A spiral model is divided into number of framework activities, also called task regions. Every framework activities represent one section of the spiral path. As the development process starts, the software team performs activities that are indirect by a path around the spiral model in a clockwise direction. It begins at the center of spiral model. Typically, there are four to eight task regions. The above figure depicts a spiral model that contains 8 task regions. Each loop in the spiral represents a phase of software process. Incremental loop might be concerned with system feasibility, next loop with the requirements definition, next system design and so on.

2.3 Spiral Incremental model

In rapidly changing world, people needs are also changing rapidly. From simple addition to online account management, people will always lack something. It is for this reason that experts from different fields working on new ideas every day. In software industry too, different programmers comes out in the open presenting what they have done. Incremental spiral model combines the capability of scheduling and staging strategy of incremental model in which the various parts of the system are developed at different times or rates, and integrated as they are completed and iterative nature with controlled, systematic aspect of spiral model.[3] Spiral incremental model support the component based software (CBS)development.

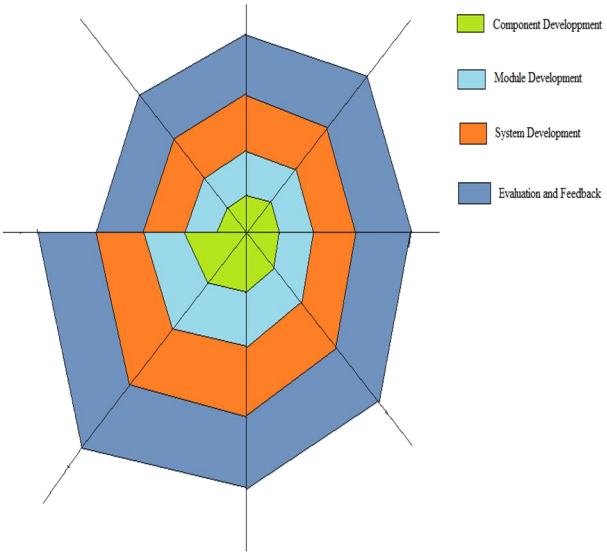

Fig. 4. Component Based Software Development

In incremental spiral model the first loop identifies the component requirement analysis then design analysis followed by developing component, modification in component, testing of component, risk analysis including verification & validation and then combining the components to get module and plan for next loop. The same process is repeated with modules in spiral loop, than modules are integrated to develop the system. So, incremental spiral model is combination of step by step incremental development of components, then components to modules and then to system with spiral looping of requirement analysis, design, development, testing, risk analysis and evaluation phases of spiral model. This model combines the capabilities of both the incremental as well as spiral model.

3. SOFTWARE REUSE

Software reuse is defined as the process of using existing software, artifacts or knowledge to build new system. It is pursued to realize benefits such as improved software quality, productivity or reliability. All software developers often proclaim: “Don’t reinvent the wheel!” But the inverse of reinvention is reuse. And reuse is often easier to proclaim than it is to achieve. Reuse is the key to progress in any area. If previously developed ideas or products were not reused then everything must be created from scratch and no progress can be made. In the development of software, everyone routinely reuse knowledge in the form of experience, processes, methods, products and tools. A reusable component may be code, but the bigger benefits of reuse come from a broader and higher-level view of what can be reused. Software specifications, designs, tests cases, data, prototypes, plans, documentation, frameworks and templates are all candidates for reuse. Software development with reuse is an approach which tries to maximize the reuse of existing software components. Benefit of this approach is that overall development costs of the software are decreased. Software reuse’s purpose is to improve software quality, productivity and reduces the development cost.[4] Software reuse is of interest because people want to build systems that are bigger and more complex, more reliable, less expensive and that are delivered on time. It is found that traditional software engineering methods inadequate and it is felled that software reuse can provide a better way of doing software engineering.[5] In the beginning of 21st century the active areas of reuse research include reuse libraries, domain engineering methods and tools, reuse design, design patterns, domain specific soft-ware architecture, component, generators, measurement and experimentation, and business and finance. Theoretically [6], up to 85% of a new application can be

developed by reusing existing software, about 65% that is domain specific and about 20% that is domain independent. Only about 15% of software that is application specific cannot be reused. The primary motivations for reuse were saving time, ensuring reliability, and saving money.[7]

3.1 RCL (Reuse Component Library)

Software reuse library consists of a repository for storing reusable assets, a search interface that allows users to search for assets in the repository, a representation method for the assets and facilities for change management and quality assessment. Much research on reuse libraries has been done as reported in the papers in the reuse roadmap.[8] Key ideas are the application of indexing methods such as free text keyword and faceted classification to reusable components. Initially, the software engineer identifies potentially reusable components from existing reusable libraries. The components are then selected, adapted and reused through composition, generalization and specialization mechanisms.[9] We can select the components that match 60% or above to our required components from RCL, the selected components are modified according to the requirement.[10] Since reuse components are more error free and reliable. One component reliability improves the overall reliability.[11] So, this will reduce the development time and increase the reliability, quality and productivity.

3.2 TCR (Tested Component Repository)

The components that are selected from RCL are modified or the components that are developed from scratch are well tested according to the project requirement. After testing these components they are stored in the TCR for further use in the software development. Testable Component Repository TCR is a testable component repository used to store the testable component which is created at the end of each phase of development for reuse and development with reuse (development after modification and development without modification) after the component testing.

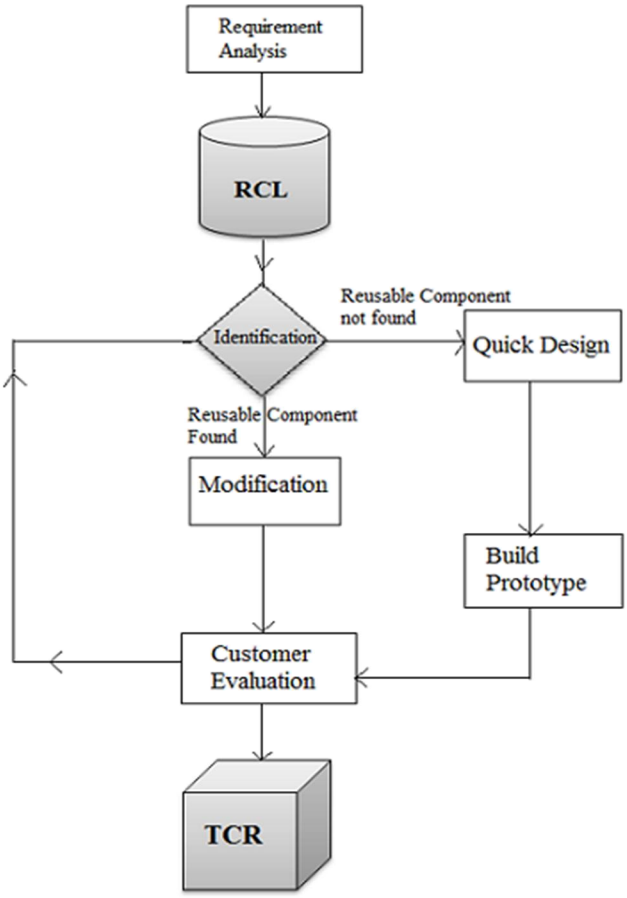

Fig. 5. TCR(Tested Component Repository)

4. PROPOSED SOFTWARE LIFECYCLE MODEL

Every software development methodology has more or less its own approach to software development. The proposed model is designed in such a way that it combines the capability of spiral and incremental model with reusability. The software development process starts with the requirement analysis and identification of required components with the help of SRS followed by component design. Required component are identified from RCL(Reuse Component Library). If the required component or similar components match 60 to 80% or more is found in RCL, then that is directly or with little modification is tested to fulfill the requirement.

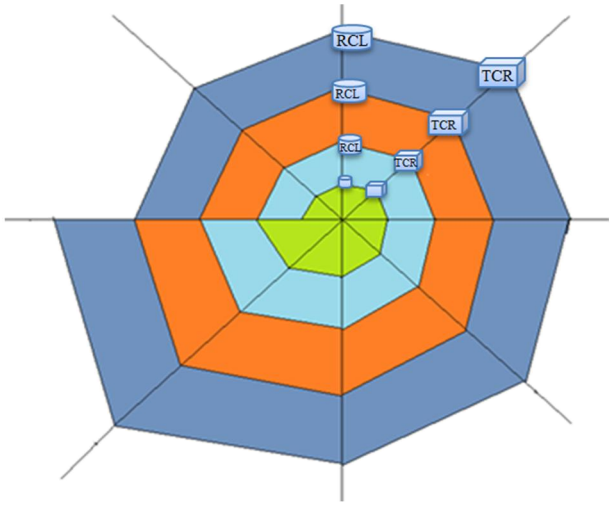

Fig. 6. (a) Spiral Incremental Reuse(SIR) Software Model

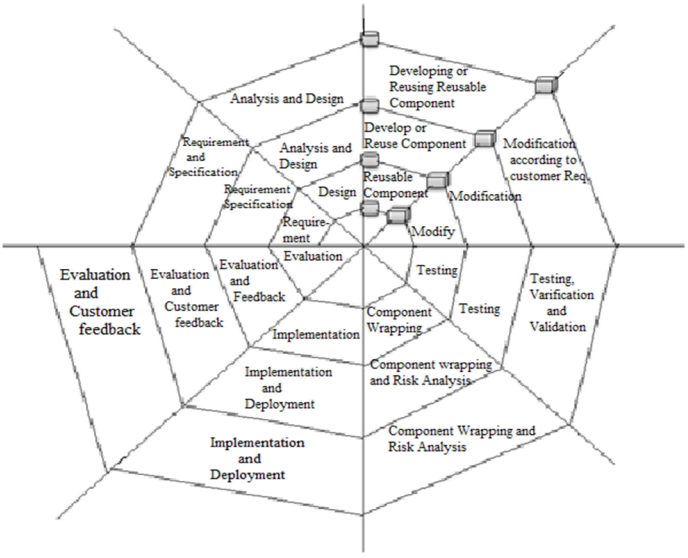

Fig. 6. (b) Spiral Incremental Reuse(SIR) Software Model

When the component is certified, it is stored in TCR (Tested Component Repository). When components are not available according to software analysis, specifications and design from RCL then start development of that component from scratch as shown in Fig.5. The component is developed using rapid prototype as in fig.5 and tested as per system requirement and then stored in TCR for further use. After component development they are wrapped and analyzed. If the component does not fulfill the SRS or customer requirement, then it again passes through the same cycle. When the given component or modules satisfies the SRS, it moves to next cycle. This reduces the risk and increases the quality. The next cycle is performed with the modules formed by wrapping components. The initial risk-identification activities are performed during every cycle. The spiral continues with the increments to form the required software project. The evaluation is done at the end of each cycle and feedback is taken to minimize the risk and increase reliability. TCR plays an important role even after the system development for maintenance and future modification in project.

4. BENEFITS OF PROPOSED SDLC

The life-cycle plan and the plan for next phase in proposed model involved a partitioning into separate activities to address management improvement, facilities improvement and development of environment for the next increment. It incorporates prototyping as a risk reduction opinion at any stage of development. In fact, prototyping and reuse risk analyses are often used in the process of going from detailed design into code. It support rework or go-backs to earlier stages as more attractive alternatives are identified or as new risk issues need resolution. The components in software development help in increasing productivity gained by reuse of design and implementation help to increase reliability by using well tested code. This help to lower the maintenance costs because of a smaller code base, help in minimizing the effects of change. Tested components also provide a well encapsulated mechanism to package, distribute and reuse software. Components are built to be used and reused in many applications. A well-executed spiral incremental reuse software development process can provide several benefits[12] like

- reduced effort in project execution
- reduces risk
- increase reliability
- rework or go back to earlier stages
- Consistent quality and reduction in defects across projects.
- Improved time to market.
- Decrease maintenance cost
- Increase the quality and productivity
- Enhanced predictability of future projects.

5. CONCLUSION

Proposed model provides a mechanism for incorporating software quality objectives into software product development.it also provide easy and faster maintenance as well as modification with the help of TCR. It appears to cover the likely phases of large software development and enforces software reusability along its phases. Moreover, it takes into account previous knowledge that software engineers may have about the application

domain, which has an influence on the prevailing approach to be followed during the software development with this model. The success of the reuse program hinges on the disciplined implementation of the proposed model. The effectiveness can be enhanced by putting in places a proper metrics plan to quantify improvement.

REFERENCES

[1] Process Models in Software Engineering Walt Scacchi, Institute for Software Research, University of California, Irvine Encyclopedia of Software Engineering, 2nd Edition, John Wiley and Sons, Inc. New York, December 2001.
[2] Sanjay Kumar, Dr C.S. Lamba Design of Model for Incremental Development of Software Modules to evaluate the Quality of Software Modules, International Journal of Modern Engineering Research (IJMER) Vol.2, Issue.4, July-Aug 2012 pp-084-088 www.ijmer.com
[3] A Comparison Between Five Models Of Software Engineering Nabil Mohammed Ali Munassar and A. Govardhan IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September 2010
[4] Modularity's impact on the quality and productivity of embedded software development: a case study in a Hong Kong company Published online: 04 Jun 2014. http://www.tandfonline.com/loi/ctqm20
[5] Sanjay Kumar et.al Software Reuse Research: Status and future. Journal of IPEM Volume 8 Issue 1 Jan14.
[6] Vikshant Khanna, Parul Mohindru Three R"s of Software Engineering: Reuse, Reengineering, Reverse Engineering. International Journal of Advanced Research in Computer Science and Software Engineering Volume 4, Issue 3, March 2014 www.ijarcsse.com
[7] Software Reuse and Reengineering: With A Case Study, Prabhot Kaur Chahal, Amritpal Singh International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-3, Issue-6, January 2014
[8] Review of Software Reuse Processes Ljupcho Antovski and Florinda Imeri IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 6, No 2, November 2013.
[9] Y: A New Component-Based Software Life Cycle Model Luiz Fernando Capretz Department of Electrical and Computer Engineering, University of Western Ontario London, Ontario, N6G 1H1, Canada Journal of Computer Science 1 (1): 76-82, 2005 http://www.researchgate.net/publication/26401710
[10] Kung-Kiu Lau et.al, The W Model for Component-based Software Development. 37th EUROMICRO Conference on Software Engineering and Advanced Applications 2011
[11] A New model for Reliability Estimation of Component-Based Software 2013 3rd IEEE International Advance Computing Conference (IACC) http://www.researchgate.net/publication/235726503
[12] Abhay Joshi A Model for Effective Asset Re-use in Software Projects, Infosys White Paper. Nov. 2007.