Comparative Analysis of Decision-making Process Model with Current Software Development Models

V. Rajiv Jetson[1] & G. Satyanarayana Prasad[2]

1 Associate Professor, Dept of CSE, Kallam Haranadhareddy Institute of Technology
2 Professor and Dean, Dept of CSE, R.V.R & J.C. College of Engineering

Abstract

For more than half a century, software has been part of modern society. With the increasing complexity of software development, the need to automate the various activities of existing software models has increased. The concept of lifecycle development software was developed, which emphasised the need to follow a structured approach to software development. To determine the success of a project, a group's software development model for software design is important. A comparative study of different existing models with the Decision Process Model (DPM), a compositionally driven software process model, is presented in this paper, in which each phase implements a 3C model, a problem-solving model. A comparative study with the Decision Process Model of common software development models shows that the DPM has important features for producing a software product that can be extended by existing components. This will help to select the best model for a particular situation.

Index Terms—*Software process model, Software product, Software development, Decision Process Model (DPM).*

1. Introduction

Developing and sustaining software systems involves a variety of highly interrelated activities [1]. A software life cycle model represents all the activities and work product necessary to develop a software system [2]. Life cycle models enable managers and developers to deal with the complexity of the process of developing software. The life cycle process model is often called software process model. Software process models are problem-solving approaches that apply requirements engineering to help solve the problems based on varying degrees of problem specification. Applying a process model and discussing its sub-processes help the team to recognize the gap between what should be and what is [3]. The software development process model that a group uses to design software is important for determining the success of a project [4].

Decision is component driven software process model, where each phase implements a problem-solving model, 3C Model [5]. These phases address what is to be built, how it will be built, building it and making it high quality. The problem solving model includes the explicit processes for technically analyzing the problem, solution space analysis, alternative analysis, dynamic design and development and scope for dynamic testing. With the problem analysis process, technical problems are identified and structured into loosely coupled sub-problems that are first independently solved and later integrated in the overall solution. In the solution space analysis process, requirements are specified using any representation and this should be refined along the software development process until the final software is delivered. In the alternative analysis process, different alternative solutions are searched and evaluated against explicit quality criteria. Dynamic design and development is component base approach, which provides scope for dynamic changes during the development life cycle. As decision process follows the component design approach, it provides scope for dynamic testing (component base testing) [6].

A number of software process models have been studied. The comparative study of Decision Process Model (DPM) existing software development approaches offers an interesting and informative mean to explicate and analyze the finest model among all.

In Section 2, we discuss the essential features of software development approaches. Then we give the detail of a selection process of software development approach for comparison in Section 3. Next, we present the comparison in Section 4 and finally, we conclude in Section 5.

2. Main Features Software Development Approaches

Various processes and methodologies have been developed over the last few decades to improve software quality, with varying degrees of success. However, it is widely agreed that no single approach can prevent project overruns and failures in all cases. This analysis leads to the main feature which is required for any software development approach to be successful. After analysis on various research papers, following are the main features which a successful software development approaches should deal with:

2.1 Requirement Specification

Requirement specification plays a vital role in software development process. It is a description of what you want the system to do. This may also be called a Business Needs Specification [7]. Software requirements change and evolve from inception to deployment. Although the majority of the requirements efforts are performed at the beginning, they often continue just before code freeze. These changes can affect both system functionality and the wider business goals of the organization for which the system is developed. If the requirements are incomplete, software developer ends up building a software product that does not meet all of the customers and user's needs [8]. In requirement specification, changes should be supported as defining clear requirements for a project can take time and lots of communication, but sometimes goals and objectives might be unclear because project sponsors lack the experience to describe what they really require [9]. However uncontrolled changes play havoc with the system under development and have caused many project failures.

In the practical world of software development, one can only come to know that the requirements are missing or misinterpreted during either QA (Quality Assurance) or UAT (User Acceptance Test) at which it is too late. Sometimes there is a disconnection between what the user says and the developer understands and aging requirements can be missed and also only know in UAT or QA, which results in project delays and cost overruns.

2.2 Changing Requirement

Requirements continuously change. This change is either because of incomplete and non-deterministic character of the requirement that is caught as a defect during the reviews or testing stage or because the needs themselves change [10]. While requirements present that dynamic nature, managing this change all through the software development lifecycle has an important impact on the success of the project. That is why when a change is needed, procedures for managing changes should be both flexible enough to allow new improvements, but also rigorous enough to prevent other product development problems [11]. In current changing needs of the software industry, software development approaches should support the changing requirements. When a change is needed in a requirement, there might be many artifacts affected by this change such as design components, test cases, and source code [12]. For these reasons it is important that changes to requirements are carefully traced, analyzed and the effects of it on the system overall is properly assessed.

Sometimes in the real world it is necessary to change requirements at the last minute as it may cause loss to the company especially in the financial world such as banking where a mistake in applying the incorrect interest rate can cause loss of millions to the company. The interest value specified in the requirements might be specified incorrectly due to human error. Adapting quickly to changing requirements can mean the difference between success and failure. It also affects the cost and time to deliver.

2.3 User Involvement

The research companies and academic institution have focused on the lack of executive support and user involvement as the two main difficulties in managing IT projects. Lack of user involvement has proved fatal for many projects. Without user involvement nobody in the business feels committed to a system and can even be hostile to it. One of the criteria, of the software project success depends on user involved from the start of the project and continuously throughout the development [13]. The developer must involve the user, as it helps in requirements elicitation and delivering all functionality of the project [14].

Sometimes the user or business sponsor is unable to provide the time that is required by some of these models for the project to be successful. Also, the user can get frustrated or lose faith in the development process due to the amount of involvement required. There needs to be a balance.

Without user involvement, a developer can be distracted by obscure features. The user then may lose their conviction in the overall project. Developer become accountable, by sharing progress openly with user on a regular basis, there is complete transparency and there is nothing to hide. This helps in timely decision making, if required due to changes.

2.4 Risk Analysis

While developing any software development product, risk is engaged. Risk analysis encourages in identifying various hazards which a software product can come across while developing it [15]. In other words, risk analysis is used to identify the high-risk elements of a project. Risk analysis is also important in the software design phase to evaluate the criticality of the system, where risks are analyzed and necessary counter measures are introduced. The purpose of risk analysis is to understand risk better and to verify and correct attributes. Every software process model must perform risk analysis.

2.5 Internal Process Control

As we know to handle the overall engineering process, it is decomposed or divided into phase. If we need to control cost, time and quality of software product efficiently than we have to control the internal process of each phase. Standard process should be followed for each phase, so as to control overall as well as internal process of software development.

2.6 Explicit Controlled Design

Once overall and internal process is controlled, then controlled design can easily be achieved. If design phase is not managed efficiently, 15% to 30 % total defects originate in this phase of software development [16]. The software development design must be fully divided into sub-problems and then the solution space should be existent for each sub-problem, later the alternative can be analyzed for each solution space. If any changes required during designing then can be handled by the software development approach without effecting much on complete designing part of the software product. The whole design activity is performed under control. This helps in achieving good quality of the product within the time frame, with all the functioning requirements of the customer.

2.7 Documentation Work

Documentation is a vital part of software engineering. The role of documentation in a software engineering environment is to communicate information to its audience and instill knowledge of the system it describes. Documentation should efficiently allow for future software development without hindering current progress, it is important in any software process model. Chapin describes several main concerns regarding documentation: quality, obsolescence or missing content. Chapin recommends to ―keeping the documentation current and trustworthy‖ [17]. It is important to note that merely being current and trustworthy do not necessarily imply applicable or useful. In a separate paper, Chapin states that as content becomes obsolete, confidence decreases and the cost of software maintenance increases [18].

Documentation is very important for maintenance, enhancement and bug fixing. If there is no or limited documentation available then it is not easy to fix bugs that are found after implementation. In fact, there is a chance that more bugs will be introduced as a result of fixing a bug in a component that is not well documented and hence, not well understood. The lack of proper documentation increases the time and cost of fixing bugs. Also, any enhancements requested, takes time to implement especially if the development team changes in personnel and there is no proper or limited documentation. Documentation is also important for the user community so that they may understand the software and its behavior. Otherwise, the users will wonder why is the software's behavior; is it a bug or as planned.

2.8 Explicit Technical Analysis of Problems

The software engineering is a problem solving process, where the software process model approaches divide the development process into various phases/activities or according to functionality. But software development process models still don't explicitly follow the technique of technically analyzing the problem [19]. The client problems may be ill-defined and include many vague requirements, but the main focus is on the precise formulation of objectives, quality criteria and the constraints for given requirement or problem.

2.9 Time Frame

The time dimension is important in software development, it is beneficial to deliver the project on given time schedule. So given time should be appropriately well thought-out. The time on task is the time the task will take to complete without interruptions, whereas duration is the time the task actually take to complete including interruptions. Using the time on task to estimate schedule is one of the common mistakes made by project managers. Long timescales for a project, led the project to fail and no longer is required by an organization. The key recommendation is that project time dimension should be short.

2.10 Cost

Usually every software development approach fits in one or the other budget limited project. It is an important point for a model, as it is best suited in which particular group. The group can be of low, intermediate and high, according to the budget. Following a wrong development approach for a particular budgeted product can lead to the cause of failure.

2.11 Type of Project Apt

Choosing the right methodology for your project is extremely important to the success of your project. Software development methodologies are like aiding tools which facilitate in smooth software development as well as implementation of software projects. Choosing wrong methodology may hamper quality of your software, budget of project, meeting project deadlines, pre-set project goals.

2.12 Flexibility of Model

Flexibility of any software development model indicates its elasticity in handling changing requirements. The requirements do change and defining clear requirements for a project can take time and lots of communication, but sometimes goals and objectives might be unclear because project sponsors lack the experience to describe what is really required. Therefore it is essential for a software development model to be flexible but with control on varying requirements in the system under development, as it may cause project failures.

2.13 Changes Incorporated

As requirements in existing software industry changes swiftly, so changes should be incorporated smoothly. Having trouble in incorporating changes will lead to delay, which further leads to failure of the project. The developer should cautiously follow and analyze changes upshot on the system, when incorporating them in the system.

2.14 Testing

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected [20]. Developer do testing of software products during development but eventually the user must run the acceptance testing to see if the system meets the business requirement. Often acceptance testing fails to catch many faults before the system goes live, as it may be due to unplanned testing, inadequately trained user who does not know the purpose of testing and insufficient time to perform testing as the project is late.

2.15 Risk Involvement

In recent years, software solutions have become extremely complex, and the complexity of software solutions is growing every day. The global market requires software with new features and capabilities developed in small time frames. The global software market demands new levels of usability, quality, reliability, and performance from software solutions. The only chance for enterprises to survive in such a market is to build software that is better than the competition, in the shortest possible time [21]. Building quality software in the terms defined by the conditions of the global software market is fundamentally hard and connected with a high level of risk [22]. A variety of highly interrelated activities involve in developing and maintaining software systems with risk associated with it. In order to manage these structured set of activities various software development models are available with varying degree of risk involved. Risks are present in every aspect of software development. Risks of software development projects must be successfully mitigated to produce successful software systems [23] [24]. Every software development project faces a significant amount of uncertainty that is usually manifested as possible risk materialization [24]. The success of a software development project is directly connected with the involved risk, i.e. project risks should be successfully mitigated in order to finish a software development project [25].

2.16 Reusability

Reusability is a measure of the ease with which one can use those previous concepts or objects in the new situations. Reusability of software has been considered as one of the most important areas for

improving software development productivity and the quality of software. Research and practice have shown that software reuse can be used for developing products from reusable assets in a routine manner, on an industrial scale [26]. Software development approaches which support effective reuse of software products is reportedly increasing productivity, saving time, and reducing cost of software development. Software can be systematically reused across the entire development life-cycle, i.e. domain analysis, requirements specification, design and implementation; it has its place even in the post-delivery stages of development, e.g. its continuing quality assessment or software maintenance [27].

2.17 Overlapping Phases

Overlapping phases means that an earlier phase in iteration does not have to completely finish before the next phase of that iteration starts. This overlapping of phases avoids the hard ―handover‖ from one phase to another and allows, for example, requirements to evolve even while the design is being done. This approach for overlapping has clear practical benefits in handling evolving requirements, it, however, does not provide any direct benefit in reducing the delivery time [28]. There is an inherent risk in overlapping, from factor such as incompatibility in the design phase, nonzero time to evolve the design specification through a series of prototyping called design iteration and incorporation of engineering change [29].

2.18 Expertise Required

Developing software requires expert specialist knowledge to be successful. Big reliable outsourcing houses have established teams of professionals trained specifically to work as a team on software development projects. An experienced team can clearly make a feasible difference when turned to, owing to effective teamwork and profound knowledge of the latest technology. A wide range of software development approaches exists, with different level of expertise required for using those approaches. The most critical resource for knowledge teams is expertise or specialized skill and knowledge but the mere presence of expertise of a team is insufficient to produce high quality work. Expertise must be managed and coordinated in order to leverage its potential [30].

2.19 Controlled Monitor

Every software development model must have an integral control monitor, which considers the environmental and problem constraints, those keep on changing throughout the development process. Control monitors keep safety inspection on changing or new requirements, to overcome the undesirable effects on the system. Each phase progress under controlled monitor and this helps in monitoring functionality and quality through proper verification and validation, using predefined qualitative and quantitative techniques.

3. Selection Process of Software Development Methodologies for Comparative Study

Many distinct variants of software development approaches are available. While going through various research papers and articles, one interesting point came into notice, that few methodologies were referred in all the papers for comparison and these methodologies could not be left-out as they are famous and highly expectable in software development industry.

Selection process started with a model which serves as a baseline for many other lifecycle models. The Waterfall is the oldest and the most well-known model in the software industry. Winston Royce in 1970 proposed the waterfall methodology to deal with the increasing complexity of aerospace software [31]. Development of the Iterative life cycle model was an effort, not to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.

Prototype model was happened to reduce inherited project risk by breaking the project into smaller segments and client involvement throughout the process. Spiral model added some form of incremental and iteration to the software development process [32]. The aim of this model was to shift the emphasis to risk evaluation and resolution. With the demand for faster software development and because of many well-documented failures of traditional Software Development Life Cycle (SDLC) models, RAD or rapid prototyping was introduced as a better way to add functionality to

an application. Subsequent incremental and iterative development approach, RUP proposed to provide guidelines, templates and tools necessary for the entire team [33].

JAD was designed to bring system developers and users of varying backgrounds and opinions together in a productive and creative environment. The meetings are the way of obtaining quality requirements and specifications [34]. Cleanroom approach was proposed to manage a quality control driven philosophy which is intended to improve the supervision and the forecast ability of the development process. In other words, it was to produce software with a certifiable level of reliability [35]. Component-based software development emerged to increase the reusability and interoperability of pieces of software. Component-based software development enables the construction of software artifacts by assembling prefabricated, configurable and independently evolving building blocks, called software components [36]. Agile development methodology provided opportunities to assess the direction of a project throughout the development lifecycle. This is achieved through regular cadences of work, known as iterations, at the end of which teams must present a potentially shippable product increment.

4. Comparison

The comparative study is performed to analyze the features of Decision Process Model and existing software development models. The Tables 1 and 2 show the comparisons of these models.

As requirements describe what is to be done but not how they are implemented [37], the activities that are performed during the requirements specifications largely focus on two areas- problem analysis and product description. In Waterfall, Spiral and Iterative models requirement specifications are performed at the beginning only, whereas in Prototype, and in Agile process the requirement specifications are frequently changed during the development process [38]. Requirement specifications are prepared in the preliminary part of the Cleanroom, RUP and CBD development approaches. In Decision Process Model (DPM) requirement specification is performed at the beginning and is updated continuously with the artifacts [6]. This helps developer to know what to build before and while development of the system in order to prevent costly rework. The later mistakes are discovered the more expensive it will be to correct them [27].

Table 1: Comparison of Software Development Approaches

Approaches -> Features	Waterfall	Prototyping	Iterative	Spiral	JAD	Decision making model
Requirement Specification	Beginning	Frequently Changes	Beginning	Beginning	Prototyped	Beginning and is updated continuously
Changing Requirement	No Support	Support	Support	Support	Support	Support
User Involvement	Only at beginning	Intermediate	Intermediate	High	High	High
Risk Analysis	Only at beginning	No	No	Yes	Yes	Yes
Internal Process Control	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Supported
Explicit Controlled Design	No	No	No	No	No	Yes

Documentation Work Req.	Vital	Weak	Weak	Yes	Limited	Yes
Explicit Technical Analysis of Problems	No	No	No	No	No	Yes
Time Frame	Long	Short	Short	Long	Medium	Least Possible
Cost	High Budget	High Budget	High Budget	High Budget	Low Budget	All Budget Type
Type of Project Apt	Big Scale	Big Scale	Big Scale	Big Scale	Small Scale	All Scale Type
Flexibility of Model	Rigid	Highly Flexible	Flexible	Flexible	Flexible	Highly Flexible
Changes Incorporated	Difficult	Easy	Easy	Medium	Medium	Easy

Testing	At End	Unit Testing	Unit Testing	Artifact testing (After each iteration)	Unit,Component,Integr. Testing	Unit,Component,Integr. Testing
Risk Involvement	High	Low	Low	Low	Medium	Low
Reusability	Limited	Weak	Weak	Yes	Yes	Yes
Overlapping Phases	No Such Phase	Yes	No	Yes	No	Yes
Expertise Required	High	Medium	Medium	High	High	Medium
Controlled Monitor	No	No	No	No	No	Yes

Table 2: Comparison of Software Development Approaches

Approaches -> Features	RAD	Cleanroom	RUP	CBD	Agile	Decision Making Model
Requirement Specification	Time Box Released	Beginning	Beginning	Beginning	Frequently Changes	Beginning and is updated continuously
Changing Requirement	Support	Support	Support	Support	Support	Support
User Involvement	Only in the beginning and after each iteration	Only in the beginning and after each increment	Beginning	Beginning	High	High
Risk Analysis	Yes but Low	No	Yes	Yes	Yes	Yes
Internal Process Control	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Support
Explicit Controlled Design	No	No	No	No	No	Yes

Documentation Work Req.	Limited	Yes	Yes	Yes	Weak	Yes
Explicit Technical Analysis of Problems	No	No	No	No	No	Yes
Time Frame	Short	Acc. To Project	Short Time Frame	Acc. to Project	Least Possible	Least Possible
Cost	Low Budget	Medium to Big Budget	Medium to Big Budget	Medium Budget	Big Budget	All Budget Type
Type of Project Apt	Small Scale	Medium to Large Scale	Medium to Large Scale	Medium Scale	Medium to Small Scale	All Scale Type
Flexibility of Model	Flexible	Medium	Medium	Flexible	Highly Flexible	Highly Flexible
Changes Incorporated	Easy	Easy	Easy	Easy	Difficult	Easy
Testing	Unit,Component,Integr. Testing	Unit,Integration, Ad-hoc Functional testing	Unit,Component,Integr. Testing	Integration Testing	Unit,Component,Integr. Testing	Unit,Component,Integr. Testing
Risk Involvement	Low	Medium	Medium	Medium	Low	Low

Reusability	Yes	No	Supp. Reusabe Existg.Classes	Supp. Reusabe Existg Classes	Low	Yes
Overlapping Phases	No	No	Yes	No	Yes	Yes
Expertise Required	Medium	Medium	Medium	Medium	Very High	Medium
Controlled Monitor	No	No	No	No	No	Yes

5. Conclusion

The results obtained from this comparison have revealed the commonalities among the different software development approaches and Decision Process Model (DPM), as well as several gaps. Firstly, all of the existing software development approaches focused on overall engineering process control but the internal control of each process is not supported. To incorporate new or changing requirements efficiently and control on cost, time and quality instead of just controlling the overall process of software engineering, each process should also be controlled. Another observation was that most of the approaches may support to the technical analysis of the problem and alternative management but not explicitly followed, which may cause the omission of these steps in software development resulting in poor quality and not as per the client/user requirements. These issues are considered in Decision Process Model.

The comparison demonstrates that the DPM has important features to produce a software product that are open for extensions of existing components and thus reusable. Overall software process as well as each phase is controlled thus provides control on cost, time and quality. It follows explicit process control, technical analysis of the problem and control design thus software product can be within the time frame, cost and quality can be raised. User/ client involvement aid developers in thinking clearly about the end product they are working on, with incorporating the new/changing requirement.

6. References

[1] P.Rajagopal, R.Lee, Thomas Ahlswede, Chia-Chu Chiang, D. Karolak, ― A New Approach for Software Requirements Elicitation‖, Proceedings of the 6th IEEE International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/ Distributed Computing, pp. 32-42, 2005.

[2] Z. Pozgaj, H. Sertic and M. Boban, ―Effective requirement specification as a precondition for successful software development project‖, IEEE Information Technology –ITI- 25th International Conference, pp. 669 – 674, 2003.

[3] C. Chittister and Y. Y. Haimes, "Risk associated with software development: A holistic framework for assessment and management", IEEE Transaction System, Man, Cybern., Volume 23, Issue 3, pp.710 -723, 1993.

[4], C.Potts, C.Richter, ―A review of the state of the practice in requirements modeling. In Proceedings International M. Lubars Symposium on Requirements Engineering‖, IEEE Computer Society, LosAlamitos, CA, pp. 2–14, 1996.

[5]Rupinder Kaur and Jyotsna Sengupta, ―Development and Analysis of 3C-Model for Software Development Lifecycle‖, IEEE 2nd International Conference on Computer Engineering and Technology (ICCET 2010), Volume 6, pp. 688 - 691, April 16 -18, Chengdu, China, 2010.

[6] Rupinder Kaur and Jyotsna Sengupta, ―A New Approach in Software Development-Decision Process Model‖, Journal of Software Engineering and Applications, Volume 3, Issue 10, pp. 998-1004, October-2010.

[7] R. D. Banker and S. A. Slaughter "The moderating effects of structure on volatility and complexity in software enhancement", Information Systems Research, Volume 11, Issue 3, pp. 219 - 240, 2000.

[8] Izak Benbasat, Iris Vessey, ―Programmer and Analyst Time/Cost Estimation‖, JSTOR-MIS Quarterly, Management Information Systems Research Center, University of Minnesota, Volume 4, Issue 2, pp. 31-43,1980.

[9] N.L.M. Noor, W.A.W. Adnan, S. Mansor, ―A survey on user involvement in software Development Life Cycle from practitioner's perspectives‖, IEEE Computer Sciences and Convergence Information Technology (ICCIT), 5th International Conference, pp. 240-243, 2010.

[10] B. Boehm and R. Ross ―Theory-W software project management: Principles and examples‖, IEEE Trans. Software Eng., Volume 15, Issue 7, pp.902 -916 1989.

[11] Don Gotterbarn and Simon Rogerson, ―Responsible Risk Analysis For Software Development: Creating The Software Development Impact Statement‖, Communications Of The Association For Information Systems, Volume 15, pp. 730-750, 2005.

[12] Suma V. and Gopalakrishna Nair T.R., ―Defect Management Strategies in Software Development‖, Research Advance In Technology, Maurizio A Strangio (Ed.), ISBN: 978-953-307-017-9, pp. 379-404, 2009.

[13] Chapin Ned., ―Trends in Preserving and Enhancing the Value of Software‖, International Conference on Software Maintenance (ICSM'00), San Jose, California, USA, IEEE Computer Society, pp. 6, 11-14 October 2000.

[14] Chapin Ned., ―Software Maintenance Types – A Fresh View‖, International Conference on Software Maintenance (ICSM'00), San Jose, California, USA, IEEE Computer Society, pp. 247, 11-14 October 2000.

[15] Jonathan Lee, ―Software Engineering with Computational Intelligence‖, Springer Publication, pp. 183-191, 2003.

[16] N.L.M. Noor, W.A.W. Adnan, S. Mansor, ―A survey on user involvement in software Development Life Cycle from practitioner's perspectives‖, IEEE Computer Sciences and Convergence Information Technology (ICCIT), 5th International Conference, pp. 240-243, 2010.

[17] Mohd Hairul Nizam Nasir and Shamsul Sahibuddin, ―Critical success factors for software projects: A comparative study‖, Scientific Research and Essays, Volume 6, Issue 10, pp. 2174-2186, 18 May, 2011.

[18] L inda Shafer, ―Software Testing for the CertifiedSoftwareDevelopmentAssociate (CSDA)‖, Developed exclusively for IEEE eLearning Library, IEEE Education & Learning, 2012.

[19] Marija Boban, Željka Požgaj, Hrvoje Sertić, ―Strategies For Successful Software Development Risk Management‖, Management, Volume 8, Issue 2, pp.77-91, 2003.

[20] E.Hall, ―Managing Risk: Methods for Software System Development‖, Addison-Wesley, New York, 1998.

[21] X.N. Lu and Ma Q.G., ―Risk analysis in software development project with owners and contractors‖, IEEE Engineering Management Conference, Volume 2, pp. 789 - 793, 2004.

[22] H. Barki , S. Rivard and J. Talbot "Toward an assessment of software development risk", J. Manag. Inf. Syst., Volume 10, Issue 2, pp.203 -225, 1993.

[23] Xiangnan Lu and Yali Ge, ―Risk analysis in project of software development‖, IEEE International Engineering Management Conference, IEMC -Managing Technologically Driven Organizations: The Human Side of Innovation and Change, pp. 72 – 75, 2003.

[24] R. Fairley and M. Willshire "Why the Vasa sank: 10 problems and some antidotes for software projects", IEEE Software Mag., Volume 20, Issue 2, pp.18 -25 2003.

[25] J.Lewi, E.Steegmans, J.De Man, ―Object-oriented approach to software development, a walk through a number of topics‖, IEEE Advanced Computer Technology, Reliable Systems and Application, 5th Annual European Computer Conference, pp. 626-633, 13-16 May, 1991.

[26] Meena Jha, ―A comparison of software reuse in software development communities‖, Software Engineering (MySEC), IEEE 5th Malaysian Conference, pp.313-318, December 13-14, 2011.

[27] DebayanBose,―ComponentBase DevelopmentApplicationinSoftware Engineering‖, Indian Statistical Institute, pp. 1-25, 2000.

[28] Jacob L. Cybulski, ―Introduction to Software Reuse‖, Department of Information Systems The University of Melbourne, Australia, Technical Report TR 96/4.

[29] Pankaj Jalote, Aveejeet Palit, Priya Kurien, ―The Timeboxing Process Model for Iterative Software Development‖, Advace in Computers, Elsevier, Volume 62, pp. 67-103, 2004.

[30] A.K.Chakravary, ―Overlapping Design and Build Cycle in Product development‖, European Journal of Operational Research, Elsevier, Volume 134, pp. 392-424, 2001.

[31] S.Faraj and L.Sproull, ―Coordinating Expertise in Software Development Team‖, Management Science, Volume 46, Issue 12, December 2000.

[32] W.W.Royce, Managing the Development of Large Software System, Proc. 9th. Intern. Conf. Software Engineering, IEEE Computer Society, 1987, 328-338 originally published in Proc. WESCON, 1970.

[33] B.Boehm, ―A Sprial Model of Software Development‖, IEEE Computer, Volume 21, Issue 5, pp. 61-72, 1988.

[34] P.Kruchten, ―Rational Unified Process – An Introduction‖, Addison-Wesley, 1999.

[35] B.Boehm, ―Software Engineering Economics‖, IEEE Transaction on Software Engineering, Volume 10, Issue 1, pp. 4-21, 1984.

[36] B.Boehm and D.Port, ―Escaping the software tar pit: model clashes and how to avoid them‖, Software Engineering Note, Volume 24, Issue 1, pp. 36-48, 1999.

[37] R.Ramsin, R.F.Paige, ―Process-Centered Review of Object-Oriented Software Development Methodologies‖, ACM Computing Surveys Volume 40, Issue 1, pp.1–89, 2008.

[38] Aaron J. Miller, ―Component-Based Software Engineering‖, Software Engineering UW-Platteville, 2012.