J. Software Engineering & Applications, 2010, 3, 998-1004
doi:10.4236/jsea.2010.310117 Published Online October 2010 (http://www.SciRP.org/journal/jsea)

A New Approach to Software Development Fusion Process Model

Rupinder Kaur, Jyotsna Sengupta

Department of Computer Science; Punjabi University, Patiala, India.
Email: rupadeo@gmail.com

Received August 12th, 2010; revised August 25th, 2010; accepted September 3rd, 2010.

ABSTRACT

There are several software process models that have been proposed and are based on task involved in developing and maintaining software product. The large number of software projects not meeting their expectation in terms of functionality, cost, delivery schedule and effective project management appears to be lacking. In this paper, we present a new software fusion process model, which depicts the essential phases of a software project from initiate stage until the product is retired. Fusion is component based software process model, where each component implements a problem solving model. This approach reduces the risk associated with cost and time, as these risks will be limited to a component only and ensure the overall quality of software system by considering the changing requirements of customer, risk assessment, identification, evaluation and composition of relative concerns at each phase of development process.

Keywords*: *Process Model, Fusion Process Model, Component Driven Development Approach,* 3*C-Model

1. Introduction

The importance of software process model for development of software product is well known, which include various steps that guide the team with common goals and strategies. Several software life cycle models or process models have appeared till now. All these models share certain characteristics. They identify stakeholder goal, specify key activities to be followed according to a certain sequence, work within time constraints and are based on what has been learned from past experiences.

The evolution of software process models has played a significant role in how models have diversified over time. Software development process and general solutions for organizing the software process belong to the standard themes of software engineering and have challenged theoreticians and practitioners for a long time. The causes of the software crisis were linked to the overall complexity of the software process and the relative immaturity of software engineering as a profession. The crisis manifested itself in several ways: projects running over-budge; projects running over-time; software was very inefficient; software was of low quality; software often did not meet requirements; projects were unmanageable and code difficult to maintain and software was never delivered.

The overall success in the development of software is still not achieved because each software development process or model consider only one or few concerns and specify a phase wise abstraction for the development, but no definite approach or model is specified for the phases of software process model. In current software engineering practices, ever changing requirements during the development process for large software development is still not managed by software process models. The solution space analysis concept of software engineering is very effective, but this concept is not completely integrated to software development yet. Alternative management, a technique which is used in mature engineering disciplines is not explicitly followed in software engineering discipline. The software development models till date follow fixed or iterative design and development approach. There is no scope for dynamic testing in software development process.

To make the software development effective and reliable, a new approach is required. Fusion process model is based on component driven development approach, which is different from component base software development. In Fusion process model, each component implements a problem solving model. It includes the explicit processes for technically analyzing the problem, solution space analysis, alternative management, dynamic design specification and development and scope for dynamic testing. In this paper, we present the new software process model which will address all the concerns and consider each phase of software development as software development process and provide an effective model for software development phases.

The main contributions of this paper are: a proposal of fusion based process model that will manage the concerns in software development and an integration of 3C-Model [1] in fusion process model for different phases that include the concept of Context (environment), Capture (Problem Solving concept for various development phases) and Control (based on environment and development constraints, quality criteria, mathematical and optimization techniques).

In Section 2, we discuss existing software development approaches with their shortcomings. Then we give a brief idea of problem solving model named 3C-Model in Section 3. Next, we present the fusion process model for effective and reliable software development in Section 4. Finally, we conclude and discuss future work in Section 5.

2. Related Approaches

Royce [2] proposed the first and most influential approach which is referred to as the waterfall model and has become the base for other models. In this approach, the whole process of software development is divided into separate process phases. Humphrey and Kellner [3] criticize the model by discussing the problems which are faced after implementing the model. These have linear structure and rigid design, rather than a dynamic problem-solving process, which would help in implementing the learning that result from user feedback and involvement. There was a change in software development approach with incremental and iterative models, also called phased development models. Graham [4] contrasted incremental development unfavorably with the fixed character of the waterfall model and suggested this approach to be for small systems only.

Boehm [5] proposed an approach which consists of a series of Waterfall-like cycles. Each cycle addresses the development of the software product at a further level of detail. Several papers indicate that for the development of software system, the identification of concerns, objective and alternatives is vital [6,7], and [8]. Later Boehm and Prasanta Bose [9] extended the spiral model to a variant called the Win-Win Spiral Model, also called win-win stakeholder approach to converge on a system's next level objectives, constrains and alternatives. It determines three milestones: life-cycle objectives, life-cycle architecture, and initial operational capability, which served as basis for software development process. This model has been formally specified and analyzed for consistency but only little is known about the correctness and usefulness of assumptions made during this process. The process and outcome of negotiations are not well defined [10].

Harlan Mills [11] proposed clean room approach, a quality control driven philosophy which is intended to improve the manageability and predictability of development process. This approach does not provide life cycle model, it provides specification for the software development. This approach does not provide life cycle model, it provides specification for the software development.

Alan Cline [12] paper work shows Joint Application Development (JAD) technique, which is an attempt to build collaboration process model. It is a technique for engaging a group or team of software developers, testers, customers, and prospective end-users in requirement specification and development of prototype. This is suitable for open source software development projects that rely on group email discussions among globally distributed users and developers [13]. J.Neighbors [14] laid stress on reusable software components guided by an application domain analysis, which is an alternative approach to developing software. Anton Jansen and Jan Bosch [15] research shows new perspective on software architecture that views software architecture as a composition of a set of explicit design decisions. This makes design decisions an explicit part of software architecture, which has become accepted concept in research and industry. The reuse model follows the component based approach, but this approach is not guided by domain analysis. It does not provide complete life cycle for software development because it considers only those systems which can be built using existing components only.

Rich Hilliard [16] gives an overview of the contributions of IEEE 1471 to the discipline of software architecture representation which fits in the theory of phased development model for different phases of software development model. The research done by Jonathan Lee [17] de-scribes the software engineering as problem solving process. Where the software process model approaches divide the development process into various phases/activities or according to functionality. But these models still don't follow the technique of technically analyzing the problem, where the technical problems are identified and divided into sub-problems that are first independently solved and later integrated in the overall solution. The client problems may be ill-defined and include many vague requirements, but the main focus is on precise formulation of objectives, quality criteria and the constraints for given requirement or problem. In technical analysis part, we can easily put this specification on each small unit of problem.

Providing a solution for a given problem is not simple, it involves the accumulation and use of huge amount of knowledge. The solution space analysis approach is still not integrated into software process models. It aims to identify the right solution domains for the given problems and extract the relevant knowledge from these domains to come up with a feasible solution. To provide quality software, it is necessary to identify the important knowledge sources for a given problem. Not all the solutions identified for a given problem are desirable. In the alternative management process, different alternative solutions are searched and evaluated against explicit quality criteria [17,18]. The high risk in software development led to the inclusion of managerial, financial and psychological factors in models [19,20], and [21]. Shaw and Garlan [22] identify seven levels of design specification capability which supports the concept of components, composition, validation, alternatives and finally automation. In the component based development, cost, time and reliability risk for an organization developing software system will shrink to component level that can be managed effectively at any stage.

Thus, a number of software process models have been studied. Most of the existing techniques manage one or more concerns of development process. From the insight gained from this study, contemporary software process models is needed, which handles various issues like requirement changes, software reuse, flexible design, user involvement and tight control over quality, cost and schedule can be overcome.

3. 3C Model

The 3C Model helps in generalizing the software development process in which a problem specification is transformed to a solution by decomposing the problem into sub-problems that are independently solved and integrated into an overall solution. This consists of multiple cycles; each cycle in 3C-Model corresponds to a transformation from one state to another, consisting of a problem specification state and a design state. The problem specification state defines the set of problems that still needs to be solved. The design state represents the tentative design solution that has been lastly defined. Initially, the design state is empty and the problem specification state includes the initial requirements. After each state transformation, a sub-problem is solved. In addition a new sub-problem may be added to the problem specification state. Each transformation process involves an evaluation step whereby it is evaluated whether the design solutions so far (design state) are consistent with the initial requirements and if there are any additional requirements identified during the evaluation. In particular, 3C-Process includes an explicit phase for searching design alternatives in the corresponding solution space and selecting these alternatives based on explicit quality criteria.

4. Fusion Process Model

Fusion is component driven software process model, where each phase implements a problem solving model. 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 fusion process follows the component design approach, it provides scope for dynamic testing (component base testing).

3C-Model assist fusion process model in generalizing the process of solving the problems in each phase. It implements component driven development approach, which provides a dynamic nature to complete software development. This makes the software development scope wider and provides firmer control over software development process. Because of the component driven approach, the risk associated with cost and time will be limited to component only and ensure the overall quality of software system, reduce the development cost and time by considering the changing requirements of customer, risk assessment, identification, evaluation and composition of relative concerns at each phase of development process. There are five fundamental phases in fusion process model and one fusion process controller to control and co-ordinate the overall development process, as shown in **Figure 1**.

4.1. Project Preparation

The project preparation phase provides the initial planning and preparation for software development project. Although each project has its own unique objectives, scope, and priorities, this phase assists in identifying and planning the primary focus areas that need to be considered. These include technical as well as project management

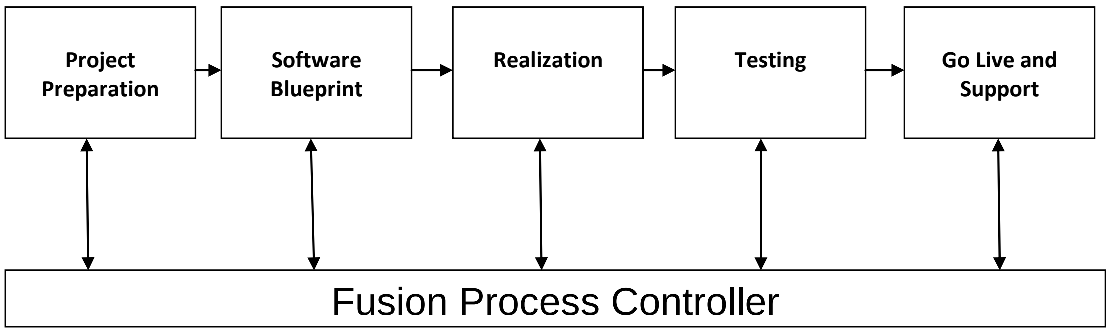

Figure 1. Fusion process model.

issues. Addressing these issues early in development will ensures that the project will proceed efficiently and establish a firm foundation for a successful development. While developing a software project, requirements definition is often considered as a one-time activity. In fact it starts with initiation of the project and is on-going activity. In feasibility analysis, requirements definition plays an important role. Every project has a feasibility analysis, regardless of the methodology used. It is essential to include important requirements definitions in feasibility analysis. When done poorly, as so often happens, the project is almost certainly destined to fail. Feasibility analysis is often referred as “project initiation” whether or not to do a project. Essentially software project team identifies what they expect the project to produce and whether it seems worthwhile to do so. If a project exists, team has made decision about it.

Extracting requirements of a desired software product is the first task in creating it. This process is called requirements elicitation. While customers probably believe they know what the software should do, it may require skill and experience in software engineering to recognize incomplete, ambiguous or contradictory requirements. Requirement analysis process provides an understanding of the client perspective of the software system. After requirements elicitation, client requirements are mapped to technical problems in the technical problem analysis process. The problem analysis process consists of the following steps:

- Generalize the Requirements: whereby the requirements are abstracted and generalized.
- Identify the Sub-Problems: whereby technical problems are identified from the generalized requirements.
- Specify the Sub-Problems: whereby the overall technical problem is decomposed into sub-problems.
- Prioritize the Sub-Problems: whereby the identified technical problems are prioritized before they are processed.

Problem reduction is a strategic approach to manage complexity. A widely known method for solving large and complex problems is to split them into simpler problems and then iteratively apply this process. The process is put into action until the sub-problems are reduced to a level of complexity at which they are easily solved or at least exhibit an irreducible level of difficulty. This paradigm for solving problems is called problem reduction. In this, a problem in a given domain is decomposed into a structured set of sub-problems in the same domain. Each sub-problem is evaluated for suitability to be further decomposed until each sub-problem is determined solvable. This problem reduction paradigm has been successfully applied to problems in a variety of application domains and in many phases of the process in which a top-down decision making strategy is applied.

The problem reduction can be expensive if not handled properly. Often, the same process must be done repeatedly for a similar type of problem with only minor differences. As a result, problem reduction may cost even more over time as problems become more complex. An important approach for handling the side effects of problem reduction is to build reusable sub-problems and solutions, instead of continually reinventing a related system reductive hierarchy. Such reusable sub-problems and solutions can be stored in a components library and retrieved as required. Complete solutions can then be obtained by using and reassembling appropriate sub-solution components.

4.2. Software Design

Architecture is established in the design phase. This phase starts with the inputs delivered by the initial phase and maps the requirements into architecture. The architecture defines the components, their interfaces and behaviors. The deliverable design document is the architecture. The design document describes a plan to implement the requirements. This phase represents the "how" phase. Details on computer programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established. The design may include the usage of existing components.

The Solution Domain Analysis process applied in software design phase aims to provide a solution domain model that will be utilized to extract the architecture de-sign solution. It consists of the following activities:

- Identify and prioritize the solution domains for each sub-problem
- Identify and prioritize knowledge sources for each solution domain.
- Extract solution domain concepts from solution domain knowledge.
- Structure the solution domain concepts.
- Refine the solution domain concepts.

4.2.1. Identify and Prioritize the Solution Domains

To the overall problem and each sub-problem, search for the solution domains are prepared that provide the solution abstractions to solve the technical problem. The solution domains for the overall problem are more general than the solution domains for the sub-problems. In addition, each sub-problem may be recursively structured into sub-problems requiring more concrete solution domains.

An obstacle in the search for solution domains is possibly the large space of solution domains which leads to a time-consuming search process. To support this process, categorizations of the solution domain knowledge into smaller sub-domains had been executed. There are different categorization possibilities. The solution domain knowledge can be categorized into application, mathematical and computer science domain knowledge etc. The application domain knowledge refers to the solution domain knowledge that defines the nature of the application, such as reservation applications, banking applications, control systems etc. Mathematical solution domain knowledge refers to mathematical knowledge such as logic, quantification, calculation and optimization techniques, etc. Computer science domain refers to knowledge of the computer science solution abstractions, such as programming languages, operating systems, databases, analysis and design methods etc.

4.2.2. Identify and Prioritize Knowledge Sources

Each identified solution domain covers a wide range of solution domain knowledge sources. These knowledge sources may not all be suitable and vary in quality. For distinguishing and validating the solution domain knowledge sources we basically consider the quality factors of objectivity and relevancy. The objectivity quality factor refers to the solution domain knowledge sources itself, and defines the general acceptance of the knowledge source. The relevancy factor refers to the relevancy of the solution domain knowledge for solving the identified technical problem.

4.2.3. Extract Solution Domain Concepts from Solution Domain Knowledge

Once the solution domains have been identified and prioritized, the knowledge acquisition from the solution domain sources can be initiated. The solution domain knowledge may include a lot of knowledge that is covered by books, research papers, case studies, reference manuals, existing prototypes/systems etc. Due to the large size of the solution domain knowledge, the knowledge acquisition process can be a labor-intensive activity, so a systematic approach for knowledge acquisition is required.

In this approach, we make a distinction between the knowledge elicitation and concept formation process. Knowledge elicitation focuses on extracting the knowledge and verifying the correctness and consistency of the extracted data. Hereby, the irrelevant data is disregarded and the relevant data is provided as input for the concept formation process. Knowledge elicitation techniques are eminent and its role in the knowledge acquisition process is reasonably well-understood. The concept formation process is mapping the extracted knowledge with technical problems. The concept formation process utilizes the knowledge and get abstract to form concept. One of the basic abstraction techniques in forming concepts is by identifying the variations and commonalities of extracting information from the knowledge sources.

4.2.4. Structure the Solution Domain Concept

The identified solution domain concepts are structured using parent-child relationship. Here all the attributes and operations associated with the concept are defined.

4.2.5. Refinement of Solution Domain Concepts

After identifying the top-level conceptual architecture, the focus is on each sub-problem and follows the same process. The refinement may be necessary if the architectural concepts have a complex structure themselves and this structure is of importance for the eventual system.

The ordering of the refinement process is determined by the ordering of the problems with respect to their previously determined priorities. Architectural concepts that represent problems with higher priorities are handled first and in the similar manner the refinement of the architectural concepts is done.

4.3. Realization

The purpose of realization phase is to develop software system for requirements based on the software design;

the team builds the components either from scratch or by composition. Given the architecture document from the design phase and the requirement document from the analysis phase, the team builds exactly what has been requested, though there is still room for innovation and flexibility.

4.3.1. Alternative Design Space Analysis

The alternative space is define as a set of possible design solutions that can be derived from a given conceptual software architecture. The alternative design space analysis aims to depict this space and consists of the two sub-processes: define the alternatives for each concept and describe the constraints. Let us now explain these sub-processes in more detail.

4.3.1.1. Define the Alternatives for each Concept

In this approach the various architecture design alternatives are derived from well-established concepts in the solution domain that have been leveraged to the identified technical problems.

4.3.1.2. Describe the Constraints

The total set of alternatives per concept may be too large and/or not relevant for solving the identified problems. Therefore, to define the boundaries of the architecture it is necessary to identify the relevant alternatives and omit the irrelevant ones.

4.4. Testing

Quality of software product is very important while developing it. Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level. It is much easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality. A customer satisfied with the quality of a product will remain loyal and wait for new functionality in the next version. 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 complete. 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 of the implementation team.

Alternatively, another approach is to delegate testing in 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 customer's environment and because systems are released with yet-to-be-discovered errors, the customer plays an important, though reluctant, role in testing.

4.5. Go Live and Support

The purpose of the Go Live and Support phase is to cut over to live productive operation and to continuously support and improve live operations. There are two distinct periods of this phase:

4.5.1. Project End

During the time when the system is first live, all issues and problems are resolved, transition to the production support team is finalized, knowledge transfer is completed, and the project is signed off.

4.5.2. Continuous Improvement

Now that the project is over, the production support team monitors the system and resolves live business process issues. Proper change management procedures are established, and ongoing end-user training is conducted. Plans are made to continuously review and improve business processes.

4.6. Fusion Process Controller

The controller part is not a phase in process model, but it is integral part of fusion process model. The controller part helps to achieve the component driven approach by listing the details of components which are added due to requirement changes or because of new requirements. By implementing Fusion Process Controller the current software development process will not be affected by changes required due to new requirements or modifications. The affected components can be taken care separately till these components matches with the current development process.

5. Conclusions

We have presented a Fusion Process Model for software development process and discussed the concept of 3C-Model for each phase of development process model. In this approach transformation of a problem specification to a solution is made by decomposing the problem into sub-problems that are independently solved and integrated into an overall solution. This process consist of multiple cycles, were each cycle transform from problem specification state to design state. After each state trans-formation, a sub-problem is solved and a new sub-problem possibly be added to the problem specification state. Every transformation process engages an evaluation step, evaluation of design state of the initial requirements is done and verifies if additional requirements identified during this step. In particular, this process includes an explicit phase for searching design alternatives in the corresponding solution space and selecting these alternatives based on explicit quality criteria. Our work has shown that how this approach helps in controlling the overall development process by implementing component based approach. Since it is component driven approach, the threat tied to cost and time will be restricted to component only, ensuring the overall quality of software product, considering the changing requirements of customer, risk assessment, identification, evaluation and composition of relative concerns at each phase of development process.

REFERENCES

[1] R. Kaur and J. Sengupta, "Development and Analysis of 3C-Model for Software Development Lifecycle," *IEEE 2nd International Conference on Computer Engineering and Technology* (*ICCET* 2010), 16-18 April 2010, Chengdu, China, pp. 688-691.

[2] W. W. Royce, "Managing the Development of Large Software System," *Proceedings 9th International Conference on Software Engineering*, *IEEE Computer Society*, USA, 1987, pp. 328-338.

[3] W. S. Humphrey and M. I. Kellner, "Software Process Modeling: Principles of Entity Process Models," Technical Report, ACM Press, New York, February 1989, pp. 331-342.

[4] D. Graham, "Incremental Development and Delivery for Large Software System," *IEEE Computers*, Vol. 25, No. 11, 1992, pp. 1-9.

[5] B. Boehm, "Software Engineering Economics," *IEEE Transaction on Software Engineering*, Vol. 10, No. 1, 1984, pp. 4-21.

[6] B. Boehm, "Anchoring the Software Process," *IEEE Transaction on Software Engineering*, Vol. 13, No. 4, 1996, pp. 79-82.

[7] B. Boehm, "A Sprial Model of Software Development and Enhancement," *IEEE Computer*, Vol. 21, No. 5, 1988, pp. 61-72.

[8] B. Boehm and D. Port, "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them," *Software Engineering Note*, Vol. 24, No. 1, 1999, pp. 36-48.

[9] B. Boehm and P. Bose, "A Collaborative Spiral Software Process Model Based on Theory W. Processing of ICSP," IEEE Press, New York, 1994.

[10] A. Egyed and B. Boehm, "Analysis of System Requirements Negotiation Behavior Patterns," 7*th Annual International Symposium Engineering*, USA, 1997, pp. 269-276.

[11] F. P. Deek, J. A. M. McHugh and O. M. Eljabiri, "Strategic Software Engineering: An Interdisciplinary Approach," Auerbach Publications, USA, 2005, pp. 31-35.

[12] A. Cline, "Joint Application Development (JAD)," 2010. http://www.carolla.com/ wp-jad.htm

[13] W. Scacchi, "Process Models in Software Engineering," Institute for Software Research, University of California, Irvine, October 2001.

[14] J. Neighbors, "The Draco Approach to Constructing Software from Reusable Components," *IEEE Transaction on Software Engineering*, Vol. 10, No. 5, 1984, pp. 564-574.

[15] A. Jansen and J. Bosch, "Software Architecture as a Set of Architecture Design Decision," *IEEE Computer Society*, 2005, pp. 109-120.

[16] R. Hilliard, "IEEE Std. 1471 and Beyond," January 2001, pp. 1-3.

[17] J. Lee, "Software Engineering with Computational Intelligence," Springer Publication, New York, 2003, pp. 183-191,

[18] X. Ferre and S. Vegas, "An Evaluation of Domain Analysis Methods," 4*th CASE/IFIP*8 *International Workshop in Evaluation of Modeling in System Analysis and Design*, 1999, pp. 2-6.

[19] A. Hamid and Madnick, "Lesson Learned from Modeling the Dynamics of Software Development," *Communication ACM*, Vol. 32, No. 12, 1989, pp. 14-26.

[20] J. Ropponen and Lyytinen, "Components of Software Development Risk: How to Address Them?" *A Project Manager Survey, IEEE Transaction on Software Engineering*, Vol. 26, No. 2, 2000, pp. 98-112.

[21] B. Boehm, "Software Engineering Economics," *IEEE Transaction on Software Engineering*, Vol. 10, No. 1, 1984, pp. 4-21.

[22] N. Medvidovic and R. M. Taylor, "A Classification and Comparison Framework from Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, Vol. 26, No. 1, 2000, pp. 70-93.