ISSN: 0974-5823 Vol. 7 No. 5 May, 2022

International Journal of Mechanical Engineering

DESIGN AND IMPLEMENT OF SOFTWARE ENGINEERING DEVELOPED PROCESS MODELS

Mohammad Asghar Ali

Research Scholar, Department of Computer Application

Dr. APJ Abdul Kalam University, Indore.

Dr. Sanjay Singh Bhadoria

Research supervisor

Associate Professor, Computer Sciences and Application

Dr. APJ Abdul Kalam University, Indore.

ABSTRACT

Analysis is done of the traditional software life cycle models that are used in the field and current software development practices. It then gives a more in-depth look at the traditional models of software evolution that are used a lot and are thought of as the best way to organize software engineering projects and technologies. There are so many things that go into making software that it's hard to think of a single process model that would work for all projects. This study, however, came up with a generalized model that could help companies make good software. A general goal for evaluation, which means to think about or think about how important it is, is shown. Examining current practices, confirming theories, exploring when the subject isn't well understood, and describing the current state of things are all part of the general evaluation goals. Evaluation helps predict the future and explain why things or sequences are taking place, so it is important to do it. It is important to know about both the software process and what the software does. This evaluation, we can figure out how to evaluate it.

KEYWORDS: Evaluation, Procedure, Software Process Models, software development practices

INTRODUCTION

There are a lot of different models for how to make software, and they're all explained in software engineering, which talks about how software changes over time. The lifecycle is all about the product, and it shows how a product goes from when it's made to when it's used and then thrown away. A software process model is an abstract representation of how the architecture, design, or definition of the software process is thought of in general terms.

In software development, process models are used to deal with issues like cost, time, and quality, as well as changes in the needs of the client. The way a software product goes through its life cycle can have a big model on a lot of different issues. It's very likely that if the process is weak, the end product will be bad as well. There has been a lot of process done in this field, but there are still not enough software process models to deal with the changes that happen during the development of large software projects. This leads to software projects that don't meet their customers' expectations in terms of functionality, cost, and delivery time. Several factors can cause a project to fail, but the most common ones are the project management process itself and how IT fits in with the culture of the project where it's being used.

Developing and maintaining software systems requires a lot of very connected work. In order to manage these structured set of activities, different models have been used over the years with a range of success. These are the Waterfall model, Continuous development, Prototyping, Spiral model, and RAD. Each product can go through different stages depending on the circumstances of each project, so there are different models to make them. For example, if the problem is well-defined and well-understood, and the user's need is almost always the same, a short waterfall-like life cycle is likely to be enough. The Waterfall Model was used a lot because it formalized some basic process control rules. It's not going to be easy to write down all the things we need to do when we don't know what the problem is or how the user wants to use it. Spiral Model: In this case, we have to go with longer and more complicated life cycles, like this one.

During each cycle of the Spiral Model, more and more of the development of the software product is looked at. They did this in a series of papers called "Win–Win Spiral Model." This is a different version of the spiral model that they came up with. The win–win stakeholder method is used to figure out three important project milestones: life-cycle architectures, life-cycle objectives, and initial operational capability. Prototyping Model helps you figure out what you need, but it can also lead to false expectations and a poorly designed system. Rapid Application Development is one of the most common ways to use the Prototyping Model (RAD).

In this model, there are strict time limits for each development phase and a lot of development on tools that make it rapid to make things happen quickly. Exploratory models use prototyping to get information about what people need. They were very simple to build, but they don't have a lot of high-level language for quick development. This model works best when very few, or no, of the system or product requirements are known ahead of time. This model is mostly based on smart guesses. This method isn't very cost-effective and sometimes leads to less-than-optimal systems, so it should only be used when there doesn't seem to be a better option.

In the Agile process model, there is less stress put on analysis and design. Implementation starts very early in the process of making software. This process model has a set amount of time. Extreme Programming (XP) was invented by Kent Beck while he was working on software. It is based on the model that you can keep improving your development over time. XP, like other agile software development, tries to cut down on the cost of change by having many short development cycles instead of one long one. It only works with teams of 12 or less people. As XP evolved, Industrial Extreme Programming (IXP) was added. It is meant to make it easier for people to work in large and spread out teams. Then, it could have a wide range of values. There is not enough evidence to show that it is good for you.

Almost all of the software projects that are done now involve some reuse of other artifacts, like design or code modules. It's called the component-based development (CBD) model, and it has many of the same features as the spiral model. It is a process that changes over time, which means that making software must be done in small steps. As a result of the component-based development model, more software can be reused, which has a number of measurable benefits for software engineers. Some people have thought about making software by putting together different models. The unified software development process is one of them. Using a combination of iterative and incremental development, the unified process defines the system's function by taking a scenario-based look at how it will work. There is a lot of focus on object oriented development.

METHODOLOGY

At the moment, the process of making software keeps changing based on what is needed and when. There are a lot of people who help with the development process. People, process, and technology all work together to make sure that software is made with the least development of resources (time, software, resources, and people) possible. If you want to be more productive, you need effective knowledge management. With the help of reuse knowledge and experience, we can do that at a very low cost. Software development is broken down into two types: traditional and agile. Traditional development is based on a plan, while agile is more flexible. In plan based or traditional development process there is lack of learning and past experience. In the agile model, learning and past experience are used to make sure users are happy, make better decisions in a changing environment, and quickly deliver a high-quality product.

Proposed hybrid model employs knowledge management which is based on knowledge flow throughout process. Figure 1 depicts the proposed model's information flow and the transitions between phases.

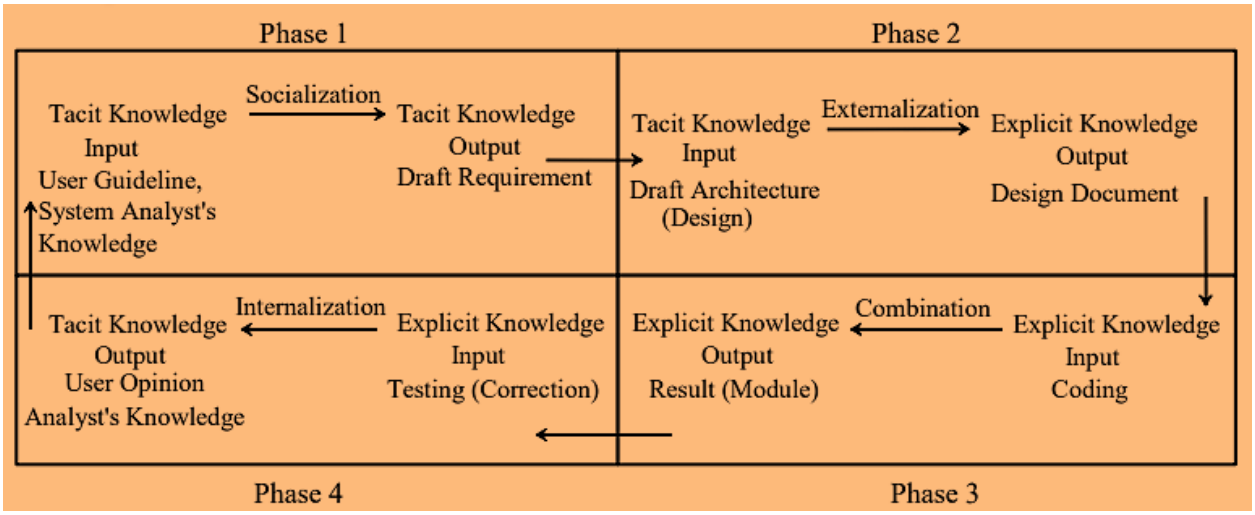

Figure 1. Proposed Method Model

Phase 1: Figure 1 shows how tacit to tacit knowledge is changed in a process called socialization. This is the first phase of the process (requirement). Phase 1 uses tacit knowledge both as an input and as an output of it. System analyst's knowledge is used as an input when gathering requirements, and after gathering requirements the draft requirement is made, as an output of this requirement

Phase 2: Phase 2 (design) is the next input after tacit to tacit knowledge is converted. This draft requirement is used as a starting point for that. This draft architecture is used as a starting input for a design document, which is an output. Phase 2 is called externalization because it uses tacit to explicit knowledge, which is how we learn.

Phase 3: It is used as an input for phase 3 (coding) and explicit to explicit knowledge conversion code to make software. This code is used as an input, and the code developer runs and checks the knowledge conversion code. That code is used as an output. In phase 3, the conversion of explicit to explicit knowledge is used, which is called a combination.

Phase 4: Use this module as input for phase 4 (testing). The software testing team looks for errors or bugs and fixes them. This module is used in phase 4 (testing). A corrected draft of the product is made and sent to the user for suggestions and feedback. This draft is used as input and sent back to the user. If there are any changes that need to be made, users and analysts meet or gather for more processing. Otherwise, this draft product is the final product. This is called internalization. In phase 4, explicit to tacit knowledge is changed, which is called internalization.

A cyclic and iterative flow of knowledge is used in the development of software. When you get to phase 4, if the user is satisfied with the draft product and its quality, then you can stop making changes to it any more. Otherwise, go back to phase 1 and start collecting user guidelines. Then, follow the same steps. The main thing about the spiral model is that it is cyclic. Each time the spiral model is used, there are four stages that it uses through in the cycle. Stage 1 is used to figure out what the goal is and what alternatives there are. Stage 2 is used to evaluate the alternative and find out what risks there are, stage 3 is used to develop and identify the next level of the product, and stage 4 is used to review the results and plan for the next iteration if necessary.

Algorithm Implementation

Stage 1: Getting software requirements by using tacit knowledge means that an expert system analyst is hired to do that job.

Stage 2: A design process is used to write down a Software Requirement Specification (SRS). When this happens, the software architecture is created with the help of tacit knowledge, which means that experienced system designers or architects are involved in that work.

Stage 3: In the next stage, with the help of a design document, software design is turned into source code using explicit knowledge, which means that new code developers can write software.

Stage 4: After each module is written, it is tested with explicit knowledge, which means that fresh software testers are used to do that work.

Stage 5: When the final product is made, it goes through people who use and test the software. If the user is satisfied with the final product and quality, there is no need for any more processing. Otherwise, go back to stage 1 and start collecting requirements.

This algorithm goes through these simple stages. It is based on iteration and cycles like the spiral model, except in stages 1 and 2 where tacit knowledge is used instead of explicit knowledge, which is what this algorithm is like. Using a hybrid model, time, money, and resources are all saved because we can reuse knowledge and experience. This can help us make software in a few iterations and improve the quality of the software, too. The proposed hybrid model uses both tacit and explicit knowledge for development, and there is less iteration to make a good product, which saves both time and money.

ANALYSIS AND RESULT

It has a big process on how software is made because of social factors, the environment, and how long you have worked on software. There are rules for the environment and management systems that make it look like you're making progress. Knowledge has its own traits and value. Both types of knowledge are important for software development, but tacit knowledge is more important because it is based on learning, experience, and creativity. The finished product has a high level of quality because experienced people are involved in the software development and there is little risk of failure or overruns of time and money. Using Table 1, you can see how the existing spiral model is different from the hybrid spiral model that we're working on.

Table 1. Comparison between Spiral model and Proposed Hybrid model

Spiral Model	Hybrid Model
A good model for larger and critical projects.	This model works with small, medium and large projects.
Focuses on risk management.	Focuses on the planning phase and risk management.
Frequent and overlapping phases.	Identifies the end point for each phase
Experience is required for its application.	Easy to understand and apply, especially with small and medium projects.
Depends on the concept of repetition to produce more than a prototype	Depends on the phases of risk analysis and tests to reveal the success of the project.

CONCLUSION

An algorithm for a hybrid model that is based on simple iteration, like this one. The hybrid model is a variation of the spiral model that uses knowledge management. Stages 1 and 2 of the proposed algorithm use tacit knowledge to make high-quality products. If we use tacit knowledge in the requirements and design stages, we get a good product.

REFERENCES

1. Garg, P.K., P. Mi, T. Pham, W. Scacchi, and G. Thunquest, The SMART approach for software process engineering, Proc. 16th. Intern. Conf. Software Engineering, 341 - 350,1994.
2. Garg, P.K. and W. Scacchi, ISHYS: Design of an Intelligent Software Hypertext Environment, IEEE Expert, 4, 3, 52-63, 1989.
3. Garg, P.K. and W. Scacchi, A Hypertext System to Manage Software Life Cycle Documents, IEEE Software, 7, 2, 90-99, 1990.
4. Goguen, J., Reusing and Interconnecting Software Components, Computer, 19,2, 16-28, 1986.
5. Graham, D.R., Incremental Development: Review of Non-monolithic Life-Cycle Development Models, Information and Software Technology, 31, 1, 7-20, January,1989.
6. Grundy, J.C.; Apperley, M.D.; Hosking, J.G.; Mugridge, W.B. A decentralized architecture for software process modeling and enactment, IEEE Internet Computing , Volume: 2 Issue: 5 , Sept.- Oct. 1998, 53 -62.
7. Grinter, R., Supporting Articulation Work Using Software Configuration Management, J. Computer Supported Cooperative Work,5, 447-465, 1996.
8. Heineman, G., J.E. Botsford, G. Caldiera, G.E. Kaiser, M.I. Kellner, and N.H. Madhavji., Emerging Technologies that Support a Software Process Life Cycle. IBM Systems J., 32(3):501-529, 1994.
9. Hekmatpour, S., Experience with Evolutionary Prototyping in a Large Software Project, ACM Software Engineering Notes, 12,1, 38-41 1987
10. Hoffnagel, G. F., and W. Beregi, Automating the Software Development Process, ,IBM Systems J.,24 ,2 1985 ,102-120 Horowitz, E. and R. Williamson, SODOS: A Software Documentation Support Environment--Its Definition, IEEE Trans. Software Engineering, 12, 8, 1986.
11. Horowitz, E., A. Kemper, and B. Narasimhan, A Survey of Application Generators, IEEE Software, 2,1 ,40-54, 1985.
12. Hosier, W. A., Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming, IRE Trans. Engineering Management, EM-8, June, 1961.
13. Humphrey, W. S., The IBM Large-Systems Software Development Process: Objectives and Direction, ,IBM Systems J., 24,2, 76-78, 1985.
14. Humphrey, W.S. and M. Kellner, Software Process Modeling: Principles of Entity Process Models, Proc. 11th. Intern. Conf. Software Engineering, IEEE Computer Society, Pittsburgh, PA, 331-342, 1989.
15. Kaiser, G., P. Feiler, and S. Popovich, Intelligent Assistance for Software Development and Maintenance, IEEE Software, 5, 3, 1988.
16. Kling, R., and W. Scacchi, The Web of Computing: Computer Technology as Social Organization, Advances in Computers, 21, 1-90, Academic Press, New York, 1982.
17. Lehman, M. M., Process Models, Process Programming, Programming Support, Proc. 9th. Intern. Conf. Software Engineering, 14-16, IEEE Computer Society, 1987.
18. Lehman, M. M., and L. Belady, Program Evolution: Processes of Software Change, Academic Press, New York, 1985
19. MacCormack, A., Product-Development Practices that Work: How Internet Companies Build Software, Sloan Management Review, 75-84,
20. Winter 2001. Mi, P. and W. Scacchi, A Knowledge Base Environment for Modeling and Simulating Software Engineering Processes, IEEE Trans. Knowledge and Data Engineering, 2,3, 283-294, 1990.