Analysis and Modeling of Change Management Process Model

S. M. Ghosh¹, H. R. Sharma¹, V. Mohabay²

¹Chhatrapati Shivaji Institute of Technology, Durg (CG) – INDIA
²Department of Electronics and IT, Kalyan Mahavidalaya Bhilai (CG) - INDIA
samghosh06@rediff.com, hrsharma44@gmail.com, mohabay@gmail.com

Abstract

As software changes is essential activity of software life cycle and hence to manage it is a vital and ambiguous task. In this paper we want to suggest a model of software change management which can focuses that change management was relevant not only in the maintenance phase, but also in the development phase of the software life cycle. The defined model does not treat software change as a post-delivery activity. With the aid of the proposed model it will be possible, in a structured way, to analyze an organization to see how well it performs on each of the specific functions of Change Management. The model can also be used to design tools or plan methods for Change Management to even to teach its concepts in a gradual and structured way. We discuss general principles for structuring the model into key areas and features and apply them to Change Management Functions. Furthermore, we discuss about the other various models available for the change management process. The ultimate goal of this work is to promote a better understanding of the process of Change Management.

Keywords: Software Engineering, Change Management, Process Model.

1. Introduction

Software is usually developed following a specific method that has some defined phases, which it goes through. Most methods have phases for analysis, design, test and implementation, after that product is released and enters the final phase of maintenance. The maintenance phase can be thought of as mini cycles of the main development phases where a new requirement (or a bug) requires a little additional analysis, a small change to the design, modifications to the implementation and updated test cases. All the products of the main cycle (specifications, design, code, test cases) are affected by changes in the maintenance phase and hence it is very important that products should be made with maintainability in mind and that Change management is able to follow changes through all phases.

One solution for achieving better control over software processes is defining processes for supporting the actual work-flows and activities taking place when people work [1]. Two main factors of successful change management are communication and control [2]. Formal and supported change management processes can be used as a tool to achieve better control over the process [3] and to support communication.

The types of processes and process models are illustrated in Figure-1 taken from Bandinelli [5]. The figure also illustrates the problems faced in understanding software processes and creating process descriptions. The two sources for examining software processes are observation of actual processes and analysis of official process descriptions. The official process descriptions are prescriptive process models, i.e. they describe how the processes should be executed. The deficiencies of the process description methods and
languages and trade-offs caused by practical limitations, such as lack of resources, change resistance, etc. result in variations between the desired process and the official process. In order to understand the actual processes in an organization the process agent aims at composing a descriptive process model, i.e. a model which describes the software processes as they are executed. The process agent faces the problems of poor visibility and traceability of the software processes. Software processes are very abstract by nature, and it is difficult to study them without introducing bias.

![Diagram of Process](image)

**Figure 1. Type of Process (Bandinelli, 1995).**

The change during the development time is not addressed at very first software process models [5] according to [7]. The models stipulated that software can be developed in successive stages, and each stage could be completed before moving to the next stage. The waterfall process model [8] introduced feedback loops between states, and recognized a maintenance phase after the delivery of the software [9][10]. Still, iteration between successive development phases was undesirable, and was considered to be caused by design errors or incomplete work made in the earlier phases. Later, when development models such as the spiral type [7] and the prototyping process model [4] emerged, the changes during the development time were considered as positive, natural phenomena by the software process models. It was recognized that effort should be directed not only to activities aiming at reducing or preventing changes from happening, but more importantly to activities supporting change management and implementation.

### 2. The Context of Change Management

There are many general texts on software development, which focuses Change management as one of the activities in the development and covers it in varying degrees. Standards and models for how to perform and manage the development process in general like ISO 9000 and Capability maturity model, do mention Change Management as an important and basic activity [17]. But then also this activity is not given that much thought and possibly new dimension to see it for better practices. Various works that has done in this context is studied and then proposed model is given as follows.
2.1 Olsen's Change Management Model

Olsen [15] presented a change management model which views the whole software development process, including both development and maintenance phases, as a dynamically overloaded queue of changes. His model views all work done by software designers as changes. A change is defined as anything that might require work to be done i.e. development of new features, filling out project management related forms, correcting errors in software, etc. Olsen's change management model is illustrated in Figure 2.

![Figure 2. Olsen Change Management Model.](image)

The model is an abstraction of the software process, where all activities are treated as changes. Therefore, it is not life cycle dependent, and can be applied in both software development and maintenance. The model does not make a distinction between different types of changes, or how different types of changes are managed. It defines the change activities on a very high level; including change creation ("Manage change" in Figure 2), change implementation ("Implement software" in Figure 2) and verification activities ("Test code" and "Inspect paper" in Figure 2). The change sources defined by the model are suggestions from the users, change proposals from verification activities, and change proposals generated by the change managers. The sponsors are treated as a source of funding and as the body monitoring the schedules.

Olsen's model points out the fact that change is a key element in software development. However, by treating all activities as changes, it loses the ability to examine and describe the specific problems and features of changing software items instead of developing new ones.

2.2 V-like Change Management Model

The V-like change model describes the technical activities for implementing a change. The process (see Figure 3) is the same for all change types. The following types of change activities are considered [13]:

![Figure 3. V-like Change Management Model.](image)
Figure 3. The V Like Change Process.

- **User support**, which includes activities for providing answers to users’ information requests and correcting misunderstandings.
- **Corrective changes**, which aim at fixing an error in software without making any changes to the requirements.
- **Evaluative changes**, which include activities for adding new functionalities in response to new or changed functional requirements.
- **Adaptive changes**, which adapt the software to changes in the operational environment.
- **Perfective changes**, which aim at improving non-functional requirements, such as execution time.
- **Preventive changes**, which improve the maintainability of the software and prevent problems in future, change activities.
- **Anticipative changes**, which anticipate future problems and aim at changing the software to be robust or easy to modify if the changes are realized in the future.

The V-like change model is designed only for the maintenance phase. It assumes that the change process takes place in an environment where the user already uses the software or the product, and the modified software has to be re-inserted to the operational environment after the change has been made. The model is generic in the organizational sense, i.e. it needs to be instantiated to the needs and requirements of the organization using it. The same model is planned to be used for all change types. However, the example given by Harjani & Queille [13] defines two variants of the model for different types of changes. The exceptional change types are emergency fixes and routine cases. The process variant for emergency fixes is followed in urgent situations. The process is optimized so that the change can be done within very tight time constraints. A simpler and lighter version of the process was proposed for routine corrective changes, where the solutions are obvious, low cost and do not have large impacts on other product parts or product operation.
The process is started by the receipt of a statement of a problem in the use of the software, or indicating a need for change. The statement can come from external sources or from the maintenance organization itself. The problem statement or change need is analyzed in the problem understanding phase. The purpose is to filter problems and determine the cause of the problem and how it should be processed. The localization activity takes care of determining precisely what is the action requested by the trigger, and which parts of the product will be affected by the change. The solution analysis step generates possible alternatives for solving the problem, and analyses their impacts. After this step, the implementation decision is made. The following alternatives for the implementation decision are given:

- Selection of one implementation alternative.
- Iteration on earlier phases is needed to find more satisfactory solution proposal
- Abandonment of the request.

If a satisfactory implementation alternative is found, the process continues into the implementation step. The implementation phase is a "mini-development" cycle. The software and associated documents, such as test data and user manuals, are changed using the normal development life cycle used in the company. The modified product is tested using regression testing techniques in order to determine if the new or modified components interact correctly with the unmodified parts, and the behavior of the product has not changed unintentionally. The acceptance procedures aim at checking that the implemented solution has solved the problem or need stated by the original change trigger. The accepted change is then closed, and the software is re-inserted into its operational environment.

2.3 Ince's Change Process Model

Ince [14] discusses how software configuration management relates to software change management. According to Ince change management covers change activities during both the software maintenance and development phases. Two main sources for changes are identified: customer requests for new features or error corrections, and the development team for problems identified in the validation phases. Other external sources, such as changes in the hardware environment and the work of product standardization or legislation bodies, or other types of internal sources, such as improvement ideas of software developers, are not discussed by Ince. He proposes that both customer and development team originated changes are managed using the change model presented in Figure 4.

The model starts with the change request, which is received by the software project. All change requests should be recorded in a change request note. Ince proposes a single change request note template for all change requests, even if the two types of changes his model covers (customer-originated new requirements and errors or problems identified either by the customer or the development team) are very different in nature and require different type of data to be recorded and analysed. All recorded changes are then submitted to the change control board. Since the process is the same for all changes, the change control board is also the same for all changes, regardless of whether the change request deals with a request for a new functionality or with a data type error identified in an inspection of the detailed design document. The change control board may decide to:

- Reject the change, in which case the change will not take place. ("Rejected" arrow in the figure)
- Batch the change, in which case the change will take place sometime in the future. ("Batched" arrow in the figure)
• Allow the change, in which case the change is implemented as soon as possible. ("Sanctioned" arrow in the figure)

When the change is sanctioned, the next step is to document the change in the configuration management documentation, and communicate it to the people responsible for its implementation. The implementation includes changing all documentation associated with the change, and producing new versions of the documents. Once the implementation has been done, the change is validated. Here Ince proposes variation according to the type of change, and suggests the size of the change to be the criterion for defining the validation strategy.

Ince's process relates the configuration management activities, which are configuration identification, configuration control, status accounting and configuration auditing, to the change process presented in Figure 4.

![Figure 4. Ince's change process model (Ince, 1994).](image)

Configuration identification is a process of specification of the components which are placed under configuration control. These components are called configuration items. A software change implies changes in component configuration items. Configuration items are identified by a version number. Configuration identification activity takes care of managing the versions and variants of the configuration items.

Configuration control activities take care of communicating the changes to the project, and inform the staff about changes to the configuration items they are dealing with. Status accounting takes care of the recording and storage of configuration data; i.e., details of configuration items and their versions and variants, list of changes made to configuration items, and recording and storage of proposed and processed changes. Configuration auditing checks that the change activities are performed as defined in the configuration management standards and procedures.

2.4 The AMES Model

The AMES model for software maintenance (see Figure 5) is primarily intended for the software maintenance phase [11]. The AMES model is mainly intended for use in the
maintenance phase of the software life cycle. At the beginning of the AMES project, the goal was to focus solely on the maintenance process, but as the work progressed, the focus shifted more to managing change in all phases of the software life cycle.

The model has three main levels: strategic, operational and service level [12].

### Layer 1: Strategic
- **Strategy definition**
- **Strategic planning**

### Layer 2: Maintenance management
- **Problem management**
- **Preparation of intervention**
- **Closure of intervention**

### Layer 3: Technical
- **Help desk**
- **Problem qualification**
- **Maintenance intervention**
- **Configuration management**

![Diagram of AMES Process Model](image)

**Figure 5. AMES Process Model.**

The AMES model supports the maintenance processes of a company. The model defines three layers: strategic, management and technical layer. The strategic layer takes care of decisions on planning the future of the product and how the customer or user relationship will be taken care of. The activities of the strategic layer include marketing, budget allocation, training and process improvement. The management layer plans, organizes and controls the actions and people who provide the change service. The management layer includes activities related to progress tracking and planning, implementation decisions, problem management and initiating and closing the change. The technical layer carries out the changes. The activities of the technical level are similar to those presented by the V-like software change model, i.e. problem understanding, localization, testing, change implementation and documentation.

### 2.5 Proposed Spiral Model for Change Management Process

Inspired by spiral model of B.Boehm [7] and the studies carried forward and analyzing the various merits and demerits of change management models, the origin of Spiral model came in existence. The proposed model was presented in Figure 6.

The spiral model divides the change management process into four cycles, in which the same main tasks are performed by each cycle, but the viewpoint is different in each cycle. The execution of the process starts from the innermost cycle. The first cycle is performed by the founder, or "owner", of the problem. The problem at this point can be either:

- A request for something new, e.g. a new feature or service
- A problem in an existing product, e.g. an error situation in a product
As a result of the first cycle, the owner of the problem decides if the problem needs to be taken care of, and how it should be taken care of. The second round of the spiral is optional. It is executed if the problem needs to be examined from a non-technical viewpoint. If the technical solution is known and clear after the first cycle, the second round can be skipped and the execution can be continued in the third round.

The third round of the process examines the modification from the system point of view, and makes an implementation plan for the last round. At this point, the affected parts of the system and the requirements for the modification task are forwarded to the fourth and final round of the process. The final round generates implements and verifies the technical solution planned during the third cycle. It also closes the modification action by delivering the result and documenting the actions and observations made.

The process is generic from the organizational point of view, i.e. it has to be instantiated for the organization using it. It is not life cycle dependent, but models the changes in both the development and maintenance phases. Different types of changes are not considered by the model.

The spiral model does not treat software change as a post-delivery activity. Ince's change management model describes only the outer cycle of the spiral model and the last quarter of the system engineering cycle, i.e. the actual technical implementation of the change. The V-model was used in defining the outermost cycle of the spiral model. Olsen's change model treats all software development activities as changes. The spiral model only addresses the actions performed for changing existing pieces of work, not creating new artifacts but modifying it based on the changes done.
2.6 Advantages of Proposed Spiral Model

The Spiral model proposed in this paper do have various advantages over other model are as follows:

1. The proposed model does not treat software change as post delivery activity. It is relevant not only in the maintenance phase but also to handle changes during the development phase of life cycle.
2. The proposed model only addresses the action performed for changing existing pieces of work, not creating new artifacts.
3. Since this model is based on Boehm model [7] hence in every cycle same task are performed, but the viewpoint is different in each cycle. Every cycle provide concrete base for the other cycle to perform better change in respect to problem understanding, evaluation of alternative solution, identifying the risks, develop the proposed change and also plan for the next phases.

2.7 Evaluation

The primary advantages of the model are that its range of option accommodates the good feature of existing software change management process model. Its risk driven approach avoid many of the difficulties. It provide good mix of benefits of all the existing approaches presented till date. The primary condition under which the proposed model had proved better than existing models are as follows:

1. If the project has low risk in such area as getting the wrong user interface or not meeting stringent performance requirement, and if it has high risk in budget and schedule predictability and control, then this risk consideration drive the proposed model to be effective.
2. If the software change requirement is very stable (implying a low risk of expensive design and code breakage due to requirements changes during maintenance ) , and if the presence of errors in the software product constitute high risk to the mission it serves, then this risk considerations drive the proposed model to reassemble precise specification and formal deductive program development.
3. If a project has a low risk in such areas as losing budget and schedule predictability and control, encountering large-system integration problems, or coping with information sclerosis, and of it has high risk in such areas as getting the wrong user interface or user decision support requirements, then this risk considerations drive the proposed model into equivalence to the evolutionary development model.

The proposed model had number additional advantages as follows:

- It focuses early attention on options involving the reuse of existing coding of software.
- It accommodates preparation of lifecycle evolution, growth, and changes of the software product.
- It provides a mechanism for incorporating software quality objectives into software product development and maintenance.
- It focuses on eliminating errors and identifies alternatives early. Since the proposed model had distributed the entire maintenance activity in four quadrants i.e. Problem Understanding, Evaluate alternative solutions, identifies and resolves risks, develop and plan for the next phase.
3. Conclusion

Software Change management is an essential discipline for enterprise IT organizations. In modern enterprises, software automates a wide variety of business processes [16]. It is very important that Change Management is planned before development starts and not just put on as an afterthought in the maintenance phase. This allows the Change Management method the possibility to influence the development discipline in a positive way. More importantly, it also permits us to gather information about the project right from the start. One of the major benefits of Change Management is that it serves as a central collection of all information. The proposed Spiral model for software change management is basically based on the study of various existence models and their merits. Since the proposed model does not involve separate approach for software development and maintenance, this aspect helps avoid the “second class citizen” status frequently associated with the maintenance phase. It also helps to avoid many of the problem that currently ensue when high risk enhancement efforts are approached in the same way as routine maintenance effort. The primary objective of this work is to provide a model that is leveled and can be used to make sure that a project has the right amount and sort of change management, neither more, nor less because in both the cases it is dangerous.

References


Authors

S.M.Ghosh
Director
Prism School of Business and Entrepreneurship
S.M.Ghosh received his Master Degree in Computer Application and authored 3 books i.e. Software Engineering, Introduction to Computing and Management Information System. Published research papers in Software Change management related topics in National and International Journals. Designed Enterprise Resource Planning Software for Job based fabrication and Casting Steel Industries. Teaching QMS, Software Engineering and MIS to the PG Students from last 12 years.

Dr. H.R. Sharma
Director, Department of Computer Science
Chhatrapati Shivaji Institute of Technology
H.R. Sharma received his M.Tech degree from Delhi University in Computer Science and has completed his Ph.D from IIT, Delhi. He has also published more than 380 research papers at National and International level. He has supervised 9 research projects sponsored by UGC & AICTE. He has more than 45 years of teaching experience at Graduate and Post Graduate level. He is presently working as the Director of Chhatrapati Shivaji Institute of Technology, Durg.

Dr. V.K.Mohabey
Head of Department, Department of Electronics and Information Technology
Kalyan Post Graduate College
V.K.Mohabey has published more than 356 research papers in which 68 are published at National level and 12 at International level. He has supervised 7 research projects sponsored by UGC, DST, MAPCOST, etc. He is presently working as a Professor and Head of Department of Electronics and Information Technology in Kalyan PG College, Bhilai. He has over 35 years of teaching experience at Graduate and Post Graduate level.