SOFTWARE DEVELOPMENT LIFE CYCLE MODEL TO ENSURE SOFTWARE QUALITY

Nihal Kececi, and Mohammad Modarres
Center for Technology Risk Studies
Department of Materials and Nuclear Engineering
University of Maryland, College Park, MD 20742, USA

ABSTRACT

In this paper, the Goal Tree Success Tree and Master Logic Diagram (GTST-MLD) is proposed to model software development life cycle to ensure software quality based on meeting the criteria for high integrity safety systems. The GTST-MLD- based software development life cycle framework allows one to (1) show how a local change affects other phases of development; (2) GTST-MLD hierarchically represent software development life cycle so as to identify missing and incomplete requirements; (3) it is easy to automate on computers, to expand and update.

1.0 Introduction

Safety-critical systems are becoming increasingly important to developers, customers and regulatory agencies. Many problems and difficulties exist in assuring safety in safety-critical computing which comes to light sometimes in the software itself and sometimes in the software development process. Many factors seem to influence the performance of software, such as, software process model, quality measurement techniques and tools, and management control methodologies. It is indeed widely accepted that the assessment of software can not be limited to verification and testing of the end product, i.e. the computer code. Other factors like the quality of the processes and management control methods also have an important impact on software performance. Several software implementations of nuclear safety systems have failed due to costly delays caused by difficulties in coordination the development and qualification process. For example, The P20 Project of Chooz B nuclear power plant [1], the shutdown system of Darlington Nuclear Power Plant [2], and the primary protection system (PPS) of Sizewell B Nuclear Power Plant [3]. The predominant belief today is that quality must be build into product in the process of development. Furthermore, the quality factors, such as safety, reliability, security i.e. that their prediction and estimation models are strongly affected to software development life cycle models.

This paper proposes a new method for software development life-cycle process. The architecture of this model is derived from the Goal Tree Success Tree and Master Logic Diagram (GTST-MLD) [4]. The elements of the architecture model are taken from IEEE Standards, which are related each life cycle phase for safety critical systems [5-16]. The proposed model is flexible for application in different software development environments and can be used in non-safety applications too. Also, In the architecture model, changing of functional and non-functional requirements is straightforward. Individual elements in the process can be updated to reflect technology advances without affecting the model itself.

2.0 Problems in the Software Quality

Software quality system is the integrated application of these three disciplines: modeling of development process (*process*), modeling of measurement of product *(product*), and modeling of management and human interactions (*people*). Understanding a discipline involves building models, testing these model and lesson to learn from real applications. High quality software developer has to deal with elements of following matrix.

Model [M]						[M*PROCESS]	[M*PRODUCT]	[M*HUMAN]
Testing [T]		Process	Product	Human	=	[T*PROCESS]	[T*PRODUCT]	[T*HUMAN]
Data [D]						[D*PROCESS]	[D*PRODUCT]	[D*HUMAN]

Main elements of software quality system are shown in Figure 1. Integration of all quality system elements requires a model. The problems to be remedied by such model are (1)-handling complexity in disciplines of quality system and theirs elements, (2) addressing some weakness of existing process models.

Complexity of development process and it's documentation, and alteration of the documentation during maintenance are important problems for improvement of quality. The documentation to be evaluated is often complex and voluminous. Because of complex relationships between technical data products, planning documentation, test requirement, and different phases of development life cycle elements, this documentation is difficult to evaluate to assure that all activities have been adequately addressed. Documentation provides communication between all groups concerned with development on the one hand, and the control of project process on the other. Schweiggert [17] notes several reasons for documentation crisis: *'Software in the application process must be constantly adapted and altered. The maintenance programmer usually does not have the time alteration to documentation. Often suitable tools are not available either. This causes the quality of documentation to suffer".* Traditional methods for verification of quality, such as checklist can fail in most complex software development processes. Audits and review can not be performed without using aids and tool, which assist in identifying compliance with standards and procedure. Moreover, complexity of development process and uncontrolled changing of process elements are negatively impact quality.

Different software development life cycles have been proposed. These have different motivations, strengths, and weakness. There is no universal life cycle model, which is considered adequate in the development environment. In traditional life cycle models, relationships between different phases of software development life cycle elements are not adequately represented and traced. It is therefore, difficult to describe the effect of any change in specified requirement on the quality, safety and objectives of software. Moreover, existing computer aids for application of process models are inflexible and difficult to handle with complexity.

In software development life cycle, identification of relationships between organization groups are important for several reasons: (1) The development process must deal with complexity and changes with requirements, test methods, technology, size etc. (2) Software faults are generated either within a phase of software

development process or at the interface between two phases. (3) Strong support from top management is a primary factor affecting the development process.

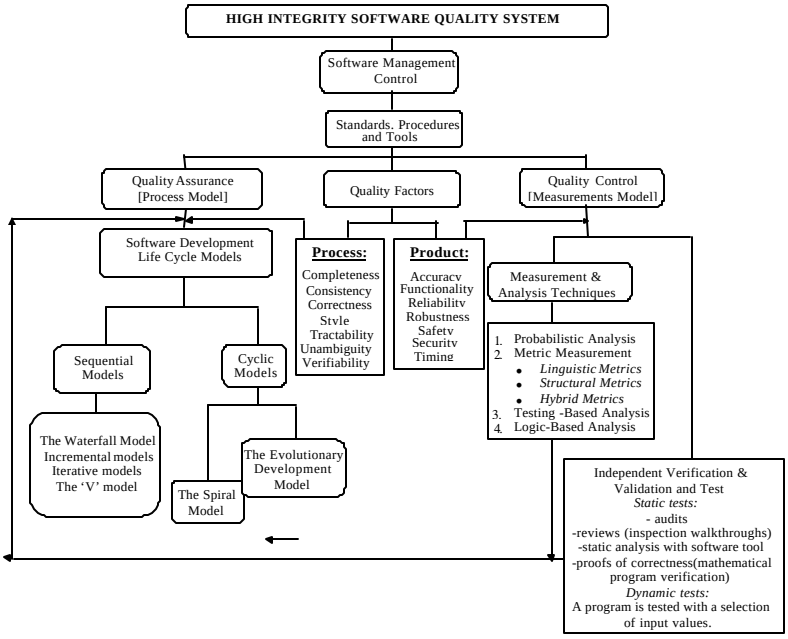

Figure 1. Elements of High Integrity Software Quality System

3.0 GTST-MLD Based Life Cycle Model Methodology

GTST-MLD framework is based on functional modeling approach to complex systems. It can easily describes and analyze interactions between system elements [4]. The concept of the GTST-MLD-based software development life cycle model follows a hierarchical decomposition of software development life cycle activities. The main step to implement a GTST-MLD-based structural hierarchy is to decompose a function into sub-elements. The decomposition process is repeated until some lowest level of elementary activities is reached. Such, one can describe the software development life cycle process at several layers. In multiple layer hierarchy, the output of first lower level can be directly linked to the inputs to other layer sub-elements. The abstraction of each hierarchy layer and its decomposition and relationships between different layers have been shown in Figure 2. Since the GTST-MLD has shown to be a powerful decomposition and analysis methodology, it is proposed that the GTST-MLD can unify representation and modeling for different phases of a traditional waterfall life cycle.

3.1 An Example: Application of the GTST-MLD Based Life Cycle Model on Nuclear Reactor Protection Systems

Software development life cycles involve complex processes using a hierarchy of activities and their documentation. Guidance on Software Review for Digital Computer-

Based Instrumentation and Control System (BTP-14) is written by US Nuclear Regulatory Commission [19]. BTP-14 was developed from IEEE Standards for different activities for the software life cycle and important design factors for safety critical software. Figure 3 was derived from the information in NUREG/CR-6101 [18] and BTP-14 [19]. Planning a software development project can be a complex process involving a hierarchic set of many activities. The result of the planning activity will be a set of documents that will be used to oversees the development of the project.

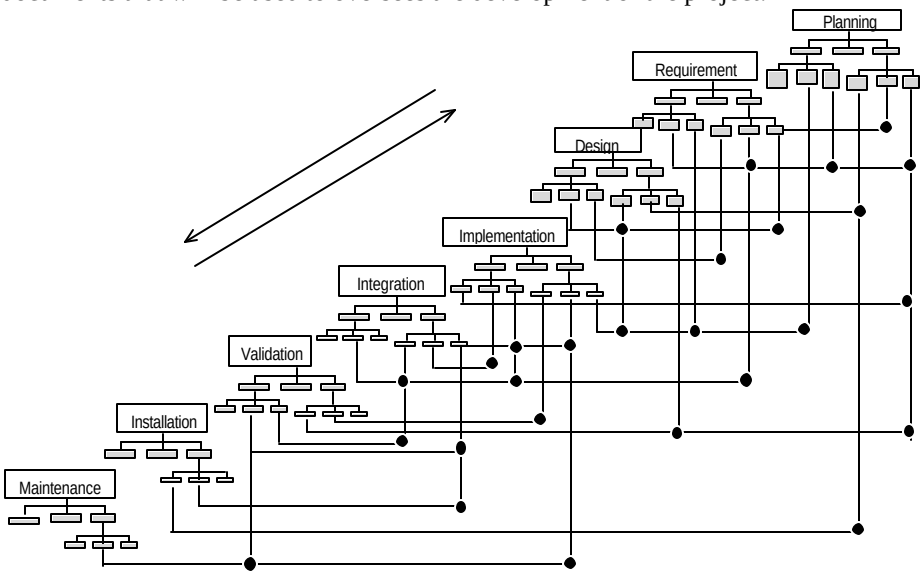

Figure 2. Abstract Presentation of GTST-MLD Based Software Development Model

In a GTST-MLD, a consistent success tree and a support lattice represent how various system activities interact with each other. The steps of implementation of a modeling are started by decomposing software project management activities and responsibilities until last hierarchical levels of development process that are operation and maintenance activities is reached. The decomposition process is repeated for each activity groups based on the relevant IEEE Standards [5-16].

The relationships between different levels of life cycle process planning have been shown in Figure 4. The connectivity relationships between nodes of two different hierarchies are classified into three categories: (1) *Overlap of responsibility;* there can be considerable overlap between the activity groups. For example, code control may reference the software configuration management plan and describe the methods by which the SQA organization will ensure that this plan follows Lawrence [18]. (2) *Support functions from bottom to top*; different level activity groups should continuously communicate each other. For example: The Risk Management is a sub-activity of project management procedure [5], sub-activity of software quality assurance (SQA) planning [7] and sub-activity of software integration marginal conditions procedure [9]. (3) *Overlap of documentation* sharing or reviewing to same documentation by different activity groups of development life cycle is often in process. For example: Safety-specific technical software documents are addressed in the “Software Development Plan” [6]. Also same documentation set are addressed in the “Software Technical Procedures”[5].

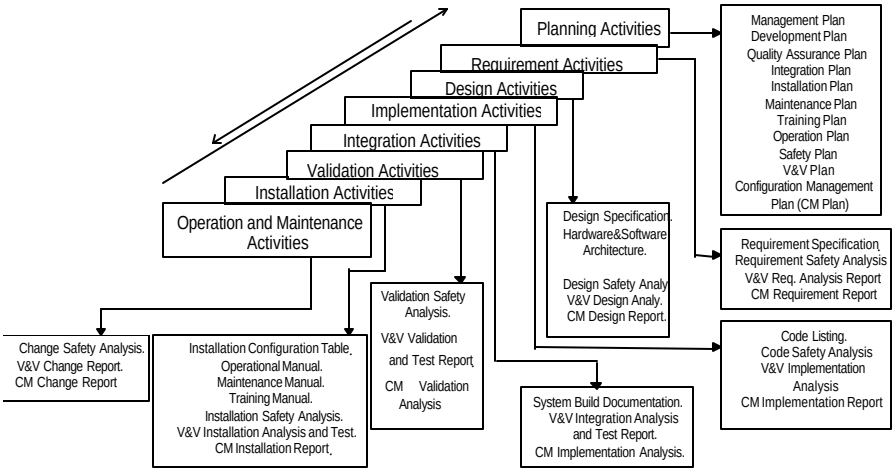

Figure 3. Software Development Life Cycle for Nuclear Protection System

The process implementation phase of development: requirement, design, code implementation, integration validation, installation, and operation and maintenance activities have complex relationships with each other's and with the planning phase. For example, in the interface between requirement and design phases, some design elements may implement more than one requirement, while other requirements may need several design elements for a successful implementation.

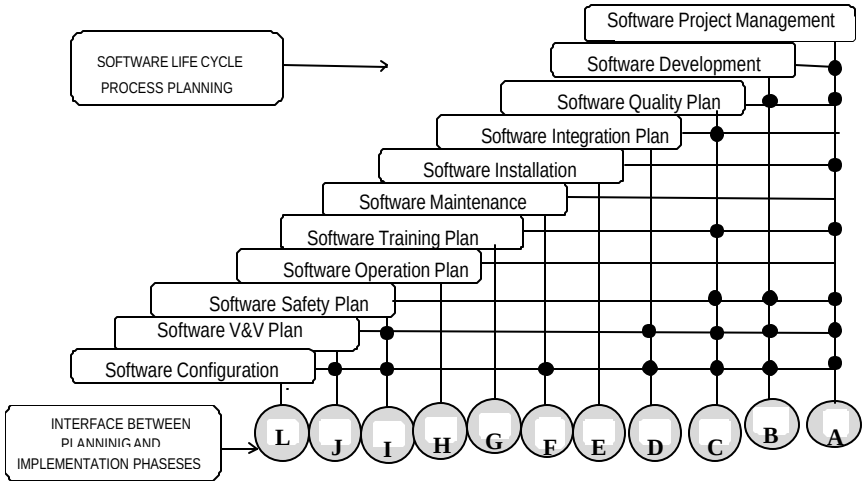

Figure 4. Relationships between Different Phases of Software Planning Activities

An analysis interaction between implementation phase elements is extremely important for safety, because, software requirement safety analysis is concerned with criticality analysis, system analysis, specification analysis and timing and sizing analysis [18]. Also, all safety-related analyses should be performed in the design and the implementation phases. Safety-related functions of systems will be easily defined after the decomposition of the functional requirements. Furthermore, model Figure 4.

Architecture will be helpful to follow up each safety-related function in the requirements, design and implementation phases.

Analysis of interactions between planning phase and implementation are important for three reasons: (1) project management will ensure that this plan is followed, (2) control of all kind changing (requirement, test case, documentation etc.) in implementation will be easy, (3) it will help the safety analyst to follow up the safety critical functions.

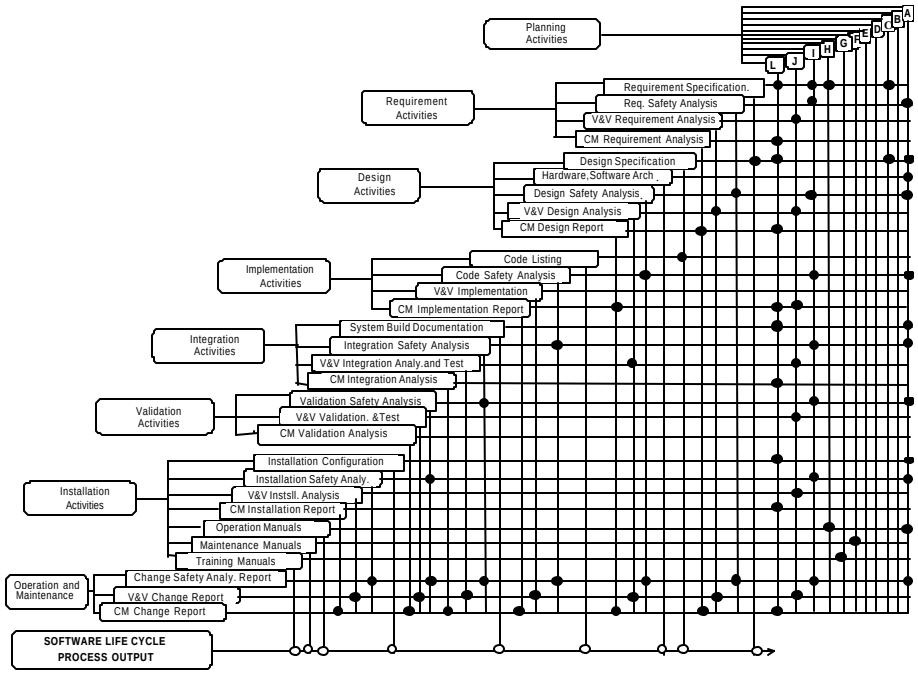

Figure 5. Relationships between Different Phase of Software Implementation

4.0 Conclusion

This paper has discussed the GTST-MLD as a new software development process model. This model has the following advantages: (1) it can easily identify missing and incomplete functional requirement. (2) Because of the tree and lattice structure, a GTST-MLD can be easily expanded to accommodate change. Therefore adding new requirement or creating new code module and changing testing plan would be an easy process. (3) The hierarchic decomposition technique can be used to decompose the complex software development process into independent modules by clear defined interfaces. It can help to determine fault generation at the interface between two phases. (4) Examination of the relationships between different phases of the software development life cycle, based on the information review will help predicting and tracing of quality factors such as: tractability, unambiguity, consistency and completeness. (5) In a GTST-MLD hierarchy, the relationship between different phases can represent including the critical functions such as; safety related, and risk related

can be identified. (6) Impact of a change on critical elements can be examined. In addition, this model can be used to assist software project management, assist software configuration management, and assist in streamlining development criteria based on relative safety critical. The model can be computerized so as to effectively follow and trace the quality issues of interest

References:

1. Appell B. Putting in a Replacement for Controbloc P20 AT Chooz B. Nuclear Eng.Int.1992; 37:45-58.
2. Craigen, D., Gerhart, S., and Ralston T. Case Study: Darlington Nuclear Generating Station. IEEE Software 1994; 11: 30-32.
3. Hughes, G., Boettcher D.B. Developments in Digital Instrumentation for Nuclear Electric's (UK) Power Plant. Nuclear Energy 1993; 32: 41-52.
4. Modarres, M. Functional Modeling of Complex Systems Using a GTST-MPLD Framework. Proceeding of the 1st International Workshop of Functional Modeling of Complex Technical Systems, Ispra, Italy 1993
5. IEEE 1058.1 IEEE Standard for Software Project Management Plan. 1987
6. IEEE 1074 IEEE Standard for Developing Software Life Cycle Process 1995.
7. IEEE 730.1 IEEE Standard for Quality Assurance Plans. 1989
8. IEEE 730.2 IEEE Guide to Software Quality Assurance Planning. 1993
9. IEC 880. Software for Computers in the Safety Systems of Nuclear Power Stations 1986.
10. IEEE 121.9 IEEE Standard for Software Maintenance 1992.
11. IEEE 1228. IEEE Standard for Software Safety Plans 1994.
12. IEEE 1012. IEEE Standard for Software Verification and Validation Plan 1986.
13. IEEE 828. IEEE Standard for Software Configuration Management Plans. 1983.
14. IEEE 1042. IEEE Guide to Software Configuration Management. 1987.
15. IEEE 830. IEEE Guide to Software Requirements Specification. 1984.
16. IEEE 1016. IEEE Recommended Practice for Software Design Descriptions. 1986.
17. Schweiggert, F., Schoitsch, E. Qualitatssicherung in der Software, OCG-Computerakademie, Seminarunterlagen. 1985.
18. Lawrence J. D. Software Reliability and Safety in Nuclear Reactor Protection Systems. NUREG/CR-6101 UCRL-ID-114839 , Lawrence Livermore National Laboratory. 1993.
19. NUREG-0800: HICB-BTP-14, Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems (Draft). 1998.