International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 24 Issue 4 – MARCH 2017.

Automatic Generation of Work Flow Model for Effective Software Development Process

S.TAMILSELVAN [#1] and V.M.NAVANEETHAKUMAR [*2]

[#] *Assistant Professor, Department Of Computer Applications, K.S.R. COLLEGE OF ENGINEERING, TN, India*

[*] *Assistant Professor, Department Of Computer Applications, K.S.R. COLLEGE OF ENGINEERING, TN, India*

***Abstract*— We design a workflow model to develop software's in an effective way, based on petrinet and other software development constraints. Our approach increases the effectiveness of the software development process by increasing the performance based on the software matrices. Project management becomes more important constraint and resource planning is also a necessary step in the process of software development. Our workflow model consists of various steps like resource planning and other steps of software development process.**

***Index Terms*— Petrinet, Reliability, COCOMO, SLIM**

I. INTRODUCTION

RPS (Reliability Prediction System) is an outline for predicting the reliability of software by taking the measures of software. Reliability Prediction can be achieved by applying multivariate analysis theory (analysis of data involving more than one variable). Ranking of existing software was done before by taking the opinions of the 30 experts and in this report it is possible by the help of the RPS framework to rank the software methods. There are few measures included and explained in this paper which helps more efficiently to calculate the reliability of the software.

There are four types of models which have been considered as potential candidates for modeling the reliability of software. These include reliability growth models, input domain models, architectural models and Beginning prediction models. (1)Reliability growth model captures failure performance during testing and generalizes its performance during procedure. Hence this category of models uses failure data and observes the failure data to derive reliability predictions. (2)Input Domain model uses properties of the input field of the software to derive correctness which approximates from test cases that executed properly. (3)Architectural models stress on the architecture of the software and derive reliability estimates by combining estimates obtained for the different modules of the software. (4) Beginning prediction model uses characteristics of the software development process from requirements to test and estimates this information to performance during operation.

A. Reliability

Reliability is probability of the non-occurrences of error. It states that an item will perform a defined method without failure during certain period of time. The numerical values of the reliability is expressed as a probability from 0 to 1and it has no units [1].Reliability is one of the validation criteria for measuring and ranking software among correlation, consistency, tracking, predictability and discriminative power. System reliability and accessibility are precise as a part of the non-functional requirements for the system. It is very important to choose an appropriate metric to specify the overall software reliability. It gives measurement by input software data and outputs a single numerical value.

B. Reliability Prediction System

Reliability prediction system describes the process which is used to estimate the constant failure rate during the useful life of software. This is one of the general forms of reliability analysis. Reliability prediction system predicts the failure rate of components and overall software reliability. These prediction system are used to calculate approximately design feasibility, evaluate design alternatives, identify possible failure areas, trade-off system design factors, and tracks reliability enhancement [2]. The impact of future proposed design of software changes on reliability is determined by comparing the reliability predictions of the existing and proposed designs of the software. The capability of the design of software to maintain an acceptable reliability level can be accessed through reliability predictions.

C. Multivariate Analysis Theory

Multivariate analysis theory consists of a set of methods that can be used when numerous measurements are made on each individual or object in one or more sample [4]. With univariate analysis, there is only one dependent variable of interest but by using multivariate analysis theory there are more than one variable involved in analysis of data. By using this theory richer realistic design of the software will be obtained. It also helps to predict the reliability and determine structure of the software.

The ranking of any software measure is predicted

On the following values:

1. The value of 1 is assign to best likely situation and hence it represents the highest reliability of any measure of the software.

2. The value of 0 is assign to worst situation and has the lowest possible reliability of any measure of the software.

3. The ranking according multivariate analysis theory can be done by predicting values lying between the first and the

last ranking criteria levels which take values between 0 and 1.

Values to be selected depends on the relative effectiveness of the ranking criteria level considered.

D. Software Quality Metrics

Software metrics is a measure of property of a piece of software or its specifications [3]. It is a quantitative measure of degree to which a system component or process have a given attribute (i.e. guess about a given attribute). There are three main categories in which metrics are classified. They are:

1) Process metrics:

This metrics deals with the activities which are related to production of software. It is mainly concerned to improve the process efficiency of the SLC.

2) Project metrics:

This metrics deals with more relevant to project team for developing software. It can be used to measure the efficiency of a project team or any other tools being used by team measures. It requires hardware, people and knowledge to measure its attribute.

3) Product metrics:

This metrics deals with the explicit results of software development activities. This requires deliverables, documentation of products used in the approach of the software product being developed.

II. BACKGROUND

Software development process becomes more time and cost constrained one. Any software development process must be a time and cost effective, the factors affecting the time and cost are the resources like man power and any other resources. To make the software development process as a time and cost effective the process of development must be planned effectively. There exist various models to develop and estimate the software but suffers with time and cost.

A. COCOMO:

It is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity. It consists of three sub models, each one offering increased fidelity along the project planning and design process. Listed in increasing fidelity, these sub models are called the Applications Composition, Early Design, and Post-architecture models. Until recently, only the last and most detailed sub model, Post-architecture, had been implemented in a calibrated software tool [Boehm Barry W, 1981]. The initial definition of COCOMO II and its rationale are described here. The definition will be refined as additional data are collected and analysed. The primary objectives of the COCOMO II effort are:

1. To develop software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's.

2. To develop software cost database and tool support capabilities for continuous model improvement.

3. To provide a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules. In priority order, these needs were for support of project planning and scheduling, project staffing, estimates-to-complete, project preparation, replanning and rescheduling, project tracking, contract negotiation, proposal evaluation, resource levelling, concept exploration, design evaluation, and bid/no-bid decisions. For each of these needs, COCOMO II will provide more upto- date support than the original COCOMO and Ada COCOMO predecessors.

B. Strategy :

The four main elements of the COCOMO II strategy are:

1. Preserve the openness of the original COCOMO;

2. Key the structure of COCOMO II to the future software marketplace sectors described above;

3. Key the inputs and outputs of the COCOMO II sub models to the level of information available;

4. Enable the COCOMO II sub models to be tailored to a project's particular process strategy.

The Application Composition Model: Suitable for projects built with modern GUI-builder tools.

The Early Design Model: Here in this model we can achieve rough estimates of a project's cost and duration before the determination of it's entire architecture. It uses a small set of new Cost Drivers, and new estimating equations. Based on Unadjusted Function Points or KSLOC.

The Post-Architecture Model: This is the most detailed COCOMO II model. This model is used after the development of overall projects architecture. It has new cost drivers, new line counting rules, and new equations.

Effort estimation: In COCOMO II effort is expressed as Person Months (PM). A person month is the amount of time one person spends working on the software development project for one month. This number is exclusive of holidays and vacations but accounts for weekend time off. The number of person months is different from the time it will take the project to complete; this is called the development schedule. For example, a project may be estimated to require 50 PM of effort but have a schedule of 11 months.

Before going to the equations we have to consider scaling factors, cost drivers andsource lines of codes.

Scale drivers: In the COCOMO II model, some of the most important factorscontributing to a project's duration and cost are the Scale Drivers. The Boehmgroup set 5 Scale Drivers to describe the project; these Scale Drivers determinethe exponent used in the Effort Equation. The 5 Scale Drivers are:

1. Precedentedness
2. Development Flexibility
3. Architecture / Risk Resolution
4. Team Cohesion
5. Process Maturity

C. SLIM (Software Life-cycle Model)

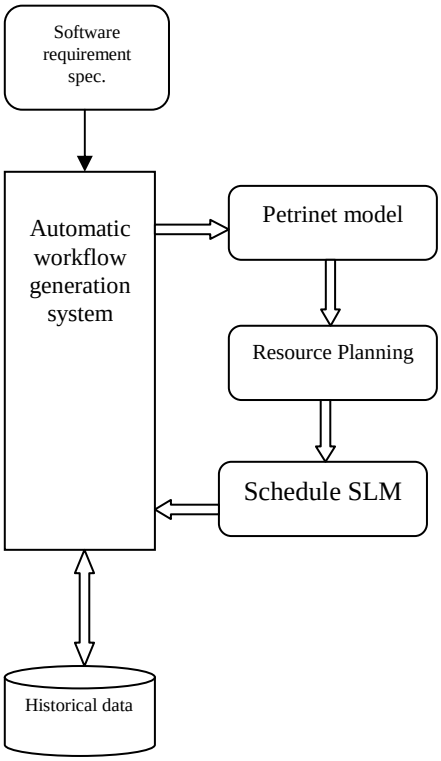
Putnam developed a constraint model called SLIM to be applied to projectsexceeding 70,000 lines of code. Putnam's model assumes that effort for softwareprojects is distributed similarly to a collection of Rayleigh curves. Putnamsuggests that staffing rises smoothly during the project and then drops sharplyduring acceptance testing. The SLIM model is expressed as two equationsdescribing relation between the

development effort and the schedule. The first equation, called the *software equation*, states that development effort is proportional to the cube of the size and inversely proportional to the fourth power of the development time. The second equation, the *manpower-buildup* equation, states that the effort is proportional to the cube of the development time.

These entire model are build earlier and the softwares developed based on these models are follow the steps as it designed and produces less effective softwares according to time and cost.

III. PROPOSED SYSTEM:

Our system generates the work flow model for the software development process according to the requirements. We submit software requirements specification using which our system produces the workflow model for the system. The work flow model consists of scheduling of resources and scheduling of all stages of the software development process.



A. *Petrinet Model:*

We maintain all the resource bundles as petrinet model. At this stage we calculate all the resources available at current time and in the future. For a project management the planning and scheduling of resources are very important and should be done in an efficient manner. Our system generates the petrinet model using all the available resources at current time and future resources. The generated petrinet model will be given to the next stage of the system.

B. *Resource Planning:*

The petrinet model generated in the previous stage is taken as the input and the resources allocated to the model are planned and the resources will be scheduled as per the petrinet model generated. All the resources assigned will be allocated at the time assigned in the model, if the resource is assigned to some other petrinet model at current time it will be ignored and will be assigned in future when it gets released from the assigned model. It's not necessary that all the resources necessary to complete the development process should be free at the time the process starts, it can be assigned when its necessary and when its freed. Our model plans the resources according to the point we specified earlier.

SDM:

Software development model consists design, coding, testing. These three steps will be iterated until the whole process gets over. In this stage using the petrinet and resource planning the process of design , coding and testing will be carried out. Once the purpose of allocated resource gets finished the resource will be released. In order to iterate the software development model the first two stages has to be rescheduled. This reduces the time and cost of the software development process.

IV. CONCLUSION:

In the earlier models like COCOMO and SLIM the resources allocated will be hold until all the processes gets finished. This makes the most of the resources in idle stage at most times. This increase the cost of software development process. In our model the work flow model is generated dynamically and the resources are assigned whenever its necessary so that it reduces the overall cost and time.

REFERENCES

[1] *[Ahituv, N., Zviran, M., Glezer, C, 1999]* Top Management Toolbox for Managing Corporate IT. Communications of The ACM 42.

[2] *[Albrecht, A.J., Gaffney,1983]* Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Transactions on Software Engineering.

[3] *[Athey, 1998]* T., Leadership Challenges For the Future. IEEE Software 15.

[4] *[Basili V, 1993]* Applying the Goal/Question/Metric Paradigm in the Experience Factory, 10th Annual CSR (Centre for Software Reliability) Workshop, Application of Software Metrics and Quality Assurance in Industry, Amsterdam, Holland.

[5] *[Bache R & Bazzana G, 1994]* Software Metrics for Product Assessment. London: McGraw-Hill

[6] *[Bersoff, 1997]* Elements of Software Configuration Management. In: Dorfman, M.,Thayer, R.H. (eds), Software Engineering. Los Alamitos: IEEE Computer SocietyPress.

[7] *[Boehm, Barry W, 1981]*, Software Engineering Economics, Prentice Hall].

[8] *[Boehm, B.W., Abts, C., Clark, B., and Devnani-Chulani, 1997]* COCOMO II Model Definition Manual. The University of Southern California. *[Boehm et al, 1997]*

[9] *[Boehm et all, 1995]* Cost Models for Future Life Cycle Processes: COCOMO 2.0.Annals of Software Engineering