Employing CoCoMo 81 for Comparing New Proposed SDLC "VISHWAS" with Existing SDLC Models

Vishwas Massey, K. J. Satao

Abstract:- Various SDLC models are available which are employed by different organizations depending upon their need and requirement of software being developed [1],[2]. Each company either follows a fixed SDLC or randomly chooses SDLC model. There were various SDLC models available but none of them were capable in addressing the issue of release management. We have developed a SDLC model – "SDLC VISHWAS" which enables the developer in handling the concept of release management along with the core SDLC phases employed for software development. We have developed software capable of generating schedules, effort, development time and staffing needed for any specified software which employs the concept of CoCoMo – 81[3],[4]. The software generates results both in text and in graphic charts which makes clear understanding for specified software being developed.

Keywords:- SDLC, CoCoMo-81, LOC, SDLC-VISHWAS, Software.

I. INTRODUCTION

There are various SDLC models that are available for developing software's. Most commonly used SDLC models are waterfall model, incremental model and spiral model. Water fall model is generally used for development of software that is small in terms of LOC (*lines of codes*) with static (*rigid*) requirements. Incremental model is similar to the waterfall model but the software is developed in increments. It is generally employed where the product's core functionality remains same but there is either change in specific functionality or addition of new functionality. We have proposed a new SDLC Model for software development engulfing the concept of release management within its core. The above model is well applicable where needs of the client is changing constantly and new features have to be added on a constant basis. Use of any suggested SDLC model entirely depends upon the resources available with the developers (*organization that is involved in the development of the software*). Due to different architecture of SDLC models, each of them leads to different LOC provided that the same software is being developed. Simply we can put this discussion as different SDLC if used for developing same software then the amount of LOC that would be coded will be different.

Manuscript received on April 26, 2012.

Vishwas Massey, M.Tech(SE) Scholar, Computer Science department, Rungta College of Engineering & Technology, Bhilai, India. (Email-vishwasmassey2yahoo.com).

Prof. K. J. Satao, Professor & HOD, CSE, IT ,& MCA Department, Rungta College of Engineering & Technology, Bhilai, India. (Email-kjsatao@rediffmail.com).

II. UNDERSTANDING LOGIC BEHIND DIFFERENT SDLC ARCHITECTURE

For keeping the discussion simple here we will be considering the most common SDLC models viz: waterfall model and incremental model. We would be discussing the architecture of these two models with the new SDLC VISHWAS. The waterfall model follows the linear sequential model in which the major SDLC phases viz: requirement analysis, design, coding, testing, and implementation are followed one after another in sequence in order to achieve a final product (*figure 1.1*) [5]. Once the final product is developed, the feedback from the end users, clients, market etc is collected which is analyzed and recorded for future developments. If the user remains unsatisfied

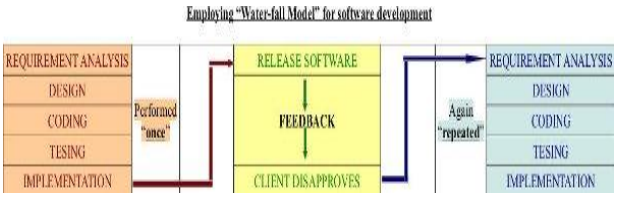

figure 1.1: Employing Waterfall model for software development

from the final product then again the SDLC phases have to be repeated and new release product is developed as a result. Thus repetition of all the SDLC phases has increased the total number of the LOC. Similarly if we discuss the architecture of the incremental model then also the similar situation arises (*figure 1.2*) [6]. SDLC phases are repeated again and again but with each repetition either there is release of somewhat a final product or without releasing the resultant software is again pipe lined into the SDLC phases. What so ever the condition arises the resultant remains the same, that is, there is increase in total number of the lines of codes.

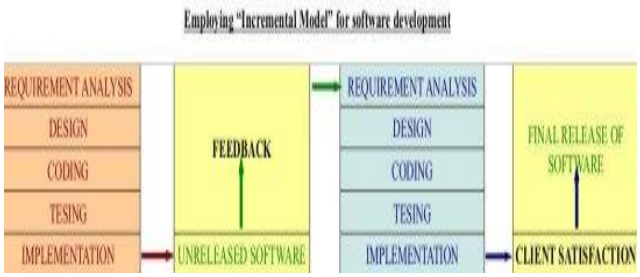

Figure 1.2: Employing Incremental model for software development

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

The new SDLC model "VISHWAS" that we had proposed eliminates this as the architecture of VISHWAS is generated in such a manner that for the final release the developer has to code minimum number of the LOCs. SDLC "VISHWAS" architecture is composed of 4 tier as developer end, release manager end, client end and the end user end (*figure 1.3*).

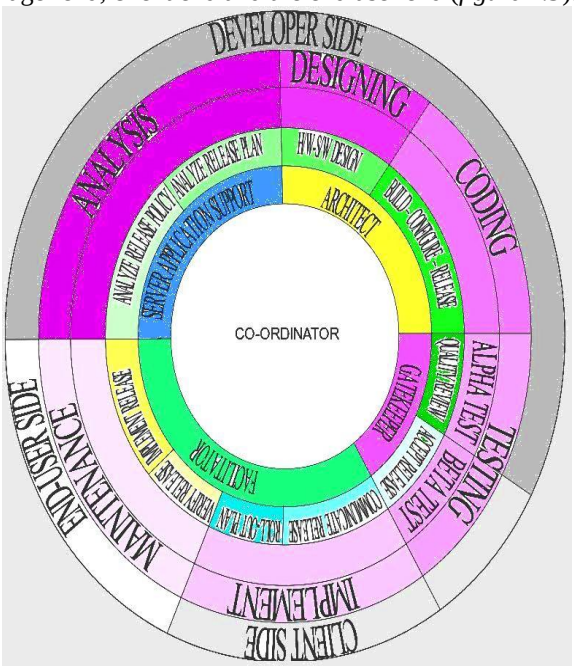

Figure 1.3: New praposed model – SDLC 'VISHWAS'

The release managers are entirely responsible for complete - immediate negotiations - communications of the need, requirement and changing functionality. Thus as soon as there is change in the need of the client, the new requirement is immediately passed on to the developer via release managers thus enabling changed functionality to be embedded in the software being developed. This helps in reducing the amount of software's codes (*minimum LOC sufficient in providing all the requirements of the client*).

III. DEVELOPING SOFTWARE

Software is developed meant for automating the work of an advocate office. Before the development, we have to understand the needs - requirements of the advocate office. Any advocate who is a regular practitioner maintains records, dairies, files etc to keep current – updated records of all his cases. The major records that are maintained by the advocate are **contact list** (*phone book for all concerned colleagues*), **record of books** (*concerned journals, law books, periodicals etc for references to be used in his cases*), **client record** (*all the concerned clients whose cases he had dealt and is currently dealing*), **case record** (*all cases either being dealt currently or had being dealt in past*). Thus on the basis of the requirements the software is developed which will satisfy the requirements and will automate the official affairs of an advocate chamber (*office*). For this particular task we have developed a software **aoa** (*advocate office automation*) which is divided into four different modules viz: security, entry, data manage and report (*figure 3.1*).

Decomposition of forms (components) of the software "aoa" into modules as:

MODULES	COMPONENT			
security	log			
entry	wlcm		mdi	
data manage	lawyer	books	clients	cases
report	report			

figure 3.1: Decomposition of forms(components) of the software "aoa" into modules

Each of the modules contains different components (*visual basic dot net forms*) in accordance to their functionality. The **security module** contains the login form which has an interface to accept user-name and password from the user. If both are correct then only the user is allowed to enter into the software else access to other modules is denied. Thus this particular module as it is called security module provides prime security to the other modules of the software as it allows authenticated and authorized user to enter into the software. Similarly other components were grouped and divided into modules. The whole software aoa was developed with the concept that all the basic needs-requirements of an advocate must be fulfilled. The modules are linked (*figure 3.2*) in such a manner that a consistent flow of control exists.

Logical flow of the software "aoa"

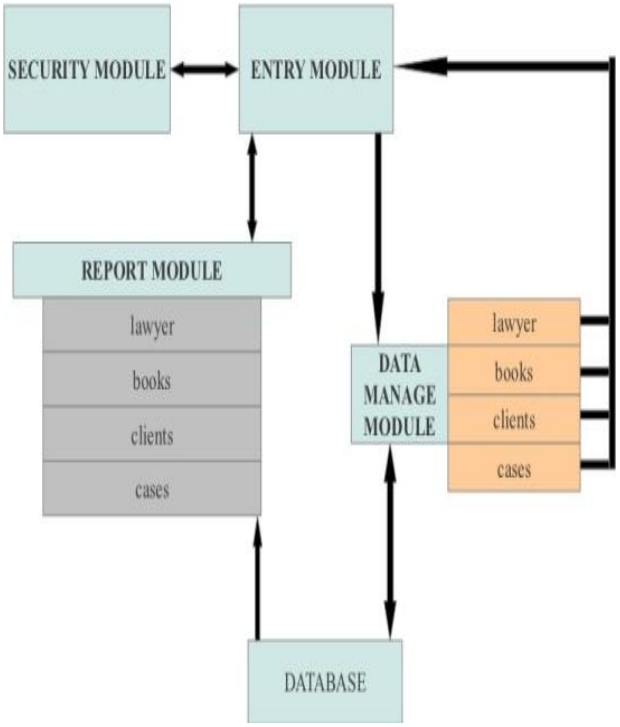

Figure 3.2: logical flow of the "aoa" software

IV. SOFTWARE DEVELOPED BY TRADITIONAL SDLC

A. *Development of software by Waterfall and Incremental model*

We have developed the above mentioned software aoa by traditional SDLC Model. Firstly we have employed "Waterfall Model" for developing the software. The software developed as a result is named as aoa_1_0.

When this software was implemented there were many reasons for rejecting by the client. Although all the aforesaid features were present in aoa_1_0 but it lacked many user-friendly features thus it was unable to satisfy the client. Now to incorporate those user-friendly features into the software another derivative of the actual software was developed as its second version. The second version of aoa_1_0 was named as aoa_1_1. This second version could have been developed using the waterfall model but then we have to start with scrap. Since architecture of waterfall model doesn't allow this we have employed another most common SDLC model.

Thus aoa_1_1 was developed by "Incremental Model". We have taken aoa_1_0 as the input to the incremental model SDLC phases and finally developed aoa_1_1. When this aoa_1_1 was subjected to tests for user acceptance then there were little lacuna discovered as compared to aoa_1_0 tests, but yes there were issues that made this software to be rejected by the client. Again aoa_1_1 was taken into the incremental model and we developed aoa_1_2 that was successful in addressing all the requirements of the client and were in accordance to his needs. Thus aoa_1_2 was able to pass all test cases successfully. This final release was termed as aoa_1 which was developed in 3 stages as aoa_1_0 followed by aoa_1_1 and again modified into aoa_1_2. The final release aoa_1 was successful in all aspects but here what the difference comes.

Say there were "i" LOC that were coded when aoa_1_0 was developed, some of the LOC say "j" were reused for developing aoa_1_1. So if there were m LOC in aoa_1_1 then total LOC for development of aoa_1_1 results to be "m-j". Similarly when aoa_1_2 was developed we again used say "x" LOC and "n" LOC from aoa_1_1 then total LOC for developing aoa_1_2 results to be "x-n". Thus we have developed the final software product aoa_1 with "i" LOC from aoa_1_0, "m+j" LOC from aoa_1_1 and "x+n" LOC from aoa_1_2. LOC in aoa_1 equals {i+(m-j)+(x-n)} which is quite high in number.

B. *Software developed by SDLC "VISHWAS"*

The same software that we had discussed above was developed by using SDLC "VISHWAS" as aoa_2 and due to its unique architecture that we have proposed in the SDLC "VISHWAS" there were no issues of rejection. Very first line of product software "aoa_2" got successfully accepted by the client. *Thus the total number of LOC that we had used in developing the software aoa_2 was much less when we compare it with the LOC needed for developing aoa_1*

V. COMPARING MODELS ON THE BASIS OF TEST CASES

To check whether each module of the software is in accordance with the client's requirement we have conducted tests. These tests have been done for each and every individual module. These test cases were generated on the basis of user-friendliness. The features of the software were needed in such a manner that the client is satisfied with the features and functionality.

Here we have discussed test cases (*figure 5.1*) that were employed for acceptance of security module. Total ten test cases were generated and the module was thoroughly checked after each release viz: aoa_1_0, aoa_1_1, aoa_1_2 and aoa_2.

Test cases generated for login page of the software

Sr. No	Acceptance Test (as per user)	aoa_1_0	aoa_1_1	aoa_1_2	aoa_2
	Developed by:	Waterfall Model	Incremental Model	Incremental Model	SDLC VISHWAS
1	Auto select text box for user-name on startup	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL	SUCCESSFUL
2	User-name – correct Password - correct	SUCCESSFUL	SUCCESSFUL	SUCCESSFUL	SUCCESSFUL
3	User-name - wrong Password - wrong	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL	SUCCESSFUL
4	User-name - correct Password - wrong	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL	SUCCESSFUL
5	User-name – wrong Password - correct	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL	SUCCESSFUL
6	User-name - blank Password - correct	UNSUCCESSFUL	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL
7	User-name - blank Password - wrong	UNSUCCESSFUL	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL
8	User-name – correct Password - blank	UNSUCCESSFUL	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL
9	User-name - wrong Password - blank	UNSUCCESSFUL	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL
10	User-name - blank Password - blank	UNSUCCESSFUL	UNSUCCESSFUL	SUCCESSFUL	SUCCESSFUL

figure 5.1: Test cases generated for login page of the software

Figure 5.2 tabulates all the test cases that were generated for each and every module of software's aoa_1_0, aoa_1_1, aoa_1_2, aoa_1 and aoa_2.

CONCLUSIVE RESULT OF aoa_1
(final version inclusive of aoa_1_0, aoa_1_1 & aoa_1_2)

Module	Form	Software	Total Design LOC	Total App. LOC	Module Design LOC	Module App. LOC	Total Test	Successful Test	Failed Test Defects
Security	log	aoa_1	4504	1914	103	62	30	16	14
Entry	wlcm				99	40	3	2	1
	mdi				181	133	33	25	8
Data Manage	lawyer				3821	1592	24	20	4
	books								
	clients								
	cases								
Report	report				300	87	15	13	2
					4504	1914	105	76	29
Total LOC			6418		6418				

Figure 5.2.1: Conclusive result of "aoa_1"

CONCLUSIVE RESULT OF aoa_1_0

Module	Form	Software	Total Design LOC		Total App. LOC		Module Design LOC		Module App. LOC		Total Test	Successful Test	Failed Test
			Reused	New	Reused	New	Reused	New	Reused	New			
Security	log	aoa_1_0	0	3261	0	1214	0	103	0	18	10	1	9
Entry	wlcm						0	99	0	15	1	0	1
	mdi						0	109	0	84	17	13	4
Data Manage	lawyer						0	2731	0	1042	16	12	4
	books												
	clients												
	cases												
Report	report						0	219	0	55	5	4	1
Total LOC			3261		1214		0	3261	0	1214	49	30	19
			4475				3261		1214				
							4475						

Figure 5.2.2: Conclusive result of "aoa_1_0"

CONCLUSIVE RESULT OF aoa_1_1

Module	Form	Software	Total Design LOC		Total App. LOC		Module Design LOC		Module App. LOC		Total Test	Successful Test	Failed Test
			Reused	New	Reused	New	Reused	New	Reused	New			
Security	log	aoa_1_1	421	1162	63	638	103	0	8	19	10	5	5
Entry	wlcm						99	0	0	25	1	1	0
	mdi						0	72	0	44	10	6	4
Data Manage	lawyer						0	1090	0	550	4	4	0
	books												
	clients												
	cases												
Report	report						219	0	55	0	5	4	1
Total LOC			1583		701		421	1162	63	638	30	20	10
			2284				1583		701				
							2284						

Figure 5.2.3: Conclusive result of "aoa_1_1"

CONCLUSIVE RESULT OF aoa_1_2

Module	Form	Software	Total Design LOC		Total App. LOC		Module Design LOC		Module App. LOC		Total Test	Successful Test	Failed Test
			Reused	New	Reused	New	Reused	New	Reused	New			
Security	log	aoa_1_2	1347	81	607	62	103	0	8	25	10	10	0
Entry	wlcm						99	0	25	0	1	1	0
	mdi						55	0	24	5	6	6	0
Data Manage	lawyer						1090	0	550	0	4	4	0
	books												
	clients												
	cases												
Report	report						0	81	0	32	5	5	0
Total LOC			1428		669		1347	81	607	62	26	26	0
			2097				1428		669				
							2097						

Figure 5.2.4: Conclusive result of "aoa_1_2"

CONCLUSIVE RESULT OF aoa_2

Module	Form	Software	Total Design LOC	Total App. LOC	Module Design LOC	Module App. LOC	Total Test	Successful Test	Failed Test Defects
Security	log	aoa_2	1462	663	124	29	10	10	0
Entry	wlcm				108	26	1	1	0
	mdi				57	30	6	6	0
Data Manage	lawyer				1090	550	4	4	0
	books								
	clients								
	cases								
Report	report				83	28	5	5	0
					1426	663	26	26	0
Total LOC			2125		2125				

Figure 5.2.5: Conclusive result of "aoa_2"

CONCLUSIVE RESULT FOR "LOC"

Module	Form	Software		Total Design LOC	Total App. LOC	Module Design LOC				Module App. LOC			
						aoa_1_0	aoa_1_1	aoa_1_2	aoa_2	aoa_1_0	aoa_1_1	aoa_1_2	aoa_2
Security	log	aoa_1	aoa_2	4504	1914	103	0	0	124	18	19	25	29
Entry	wlcm					99	0	0	108	15	25	0	26
	mdi					109	72	0	57	84	44	5	30
Data Manage	lawyer			1426	663	2731	1090	0	1090	1042	550	0	550
	books												
	clients												
	cases												
Report	report					219	0	81	83	55	0	32	28
Total LOC				6418		3261	1162	81	1426	1130	638	62	663
						4504				1914			
						6418							
				2125		2125							

Figure 5.2.6: Conclusive result of LOC

CONCLUSIVE RESULT FOR "TEST CASES"

Module	Form	Total Test				Successful Test				Failed Test			
		aoa_1			aoa_2	aoa_1			aoa_2	aoa_1			aoa_2
		aoa_1_0	aoa_1_1	aoa_1_2		aoa_1_0	aoa_1_1	aoa_1_2		aoa_1_0	aoa_1_1	aoa_1_2	
Security	log	10	10	10	10	1	5	10	10	9	5	0	0
Entry	wlcm	1	1	1	1	0	1	1	1	1	0	0	0
	mdi	17	10	6	6	13	6	6	6	4	4	0	0
Data Manage	lawyer	16	4	4	4	12	4	4	4	4	0	0	0
	books												
	clients												
	cases												
Report	report	5	5	5	5	4	4	5	5	1	1	0	0
Total		49	30	26	26	30	20	26	26	19	10	0	0
		105				76				29			

Figure 5.2.7: Conclusive result for test cases

VI. COMPARING MODELS ON THE BASIS OF LOC

We had already seen in the above discussion that number of LOC coded in much higher for aoa_1 as compared to aoa_2. But why we are emphasizing on LOC[6],[7]?LOC is the sole factor in deciding the related cost of development for any software. The CoCoMo – 81[7] is the model that helps in estimating effort (needed for software development in terms of persons-month), development time (total time that would be taken in developing the software in months), staffing needs (total number of developers that must be engaged for development, counted as persons) for any software solely on the basis of LOC coded.

Total LOC = LOC (design) + LOC (application)

We have counted the LOC needed for development of software's aoa_1 and aoa_2 and the results have been laid down in the table as in figure5.2 . We have used the above discussed concept of counting LOC, for aoa_1 we have counted LOC as {i+(m-j)+(x-n)} (which has been discussed in above section).

VII. GENERATING RESULTS FOR THE ABOVE DISCUSSION

The above discussion is of theoretical nature that has been automated in much simpler and easily understandable computer generated charts and reports. For automating the results we have developed software which we had named as "cea" (cost effort analyzer). The software has an interface that allows the user to generate effort, development time, and staffing need entirely on the basis of CoCoMo – 81. Schedules for the whole software as well as for individual modules are developed. We have allowed entry of test case results for individual modules that is summarized in a very concise form. We had employed all the standards of IT industry for the same. We had considered one working month of 19 days instead of 28 days taken as standard calendar month. This idea was even suggested by Dr. Barry W Boehm while proposing CoCoMo Model. We have generated schedules for each modules and software's on the basis of standard theoretical time that is allotted for each of the SDLC task (SDLC phases in general is specifically divided into different tasks)[8],[9],[10].

VIII. RESULTS

Finally we have developed two softwares aoa_1 and aoa_2 employing two different SDLC models. For aoa_1 we had employed waterfall model and incremental model and for aoa_2 we have used the new SDLC model "VISHWAS". The above two software's are completely identical in their functioning and features but LOC needed for their development varies.

figure 7.1: generation of effort, development and staff

On the basis of LOC we had generated results employing the software "cea". This software "cea" generates schedules, effort, development time and staffing needs for individual software by employing CoCoMo – 81 model. On the basis of LOC we have generated effort, development time and staffing need (figure 7.1).

Effort = a*(KLOC)^b *persons-month*
where a = 2.4 and b = 1.05 (*since the software being developed is of organic type*)
Development Time = c*(Effort)^d *months*
where c = 2.5 and b = 0.38 (*since the software being developed is of organic type*)
Staffing = (Effort/Development Time) *persons*

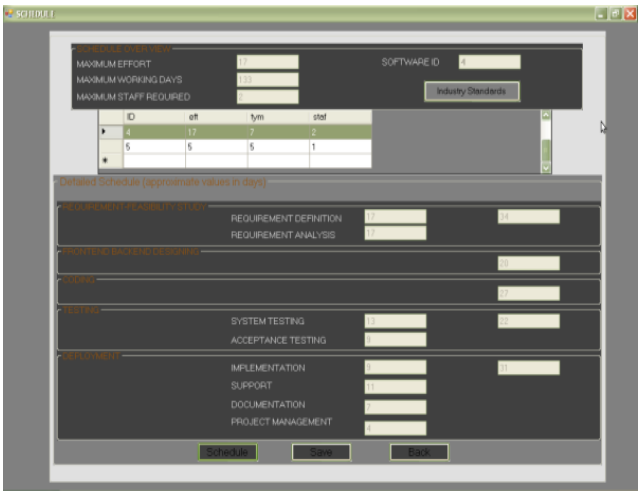

figure 7.2: generation of schedules on the basis of development time

On the basis of total development time we have generated schedule (figure 7.2) considering the standard time division (figure 7.3) for each activity commonly stated in the IT industries as we have generated the ideological time that must be given for each and every specified activity necessary for developing the concerned software. Major SDLC phases is divided into more specific sub-phases.

SDLC Task	Task	Time	In general
Analysis	Requirement Definition	10	25
	Requirement Analysis	12	
Design	Design	15	15
Coding	Coding	23	20
Testing	System Test	10	15
	Acceptance Test	6.5	
Deploy	Document	5	25
	Implement	7	
	Support	8.5	
	Project Management	3	
Total		**100**	**100**

Figure 7.3: Ideal breakup of total development time

Similarly we have generated schedule for each module and test case results (figure 7.4) on the basis of total test conducted for specific module out of which how many were successful and how many failed, finally generating defect rate. On the basis of individual module test cases we generate a concise test case summary for the whole software displaying total number of modules, summation of total test cases considering test cases performed for each module (figure 7.5). Hence we provide concise report for total tests, total successful tests, total failed tests and defect rate (total failures against total test conducted) for the whole software inclusive of all modules.

Figure 7.4: Schedules for modules and tested cases

Figure 7.5: total generated cases

The above results could be summarized either in form of textual reports (figure 7.6) or graphical charts (figure 7.7)

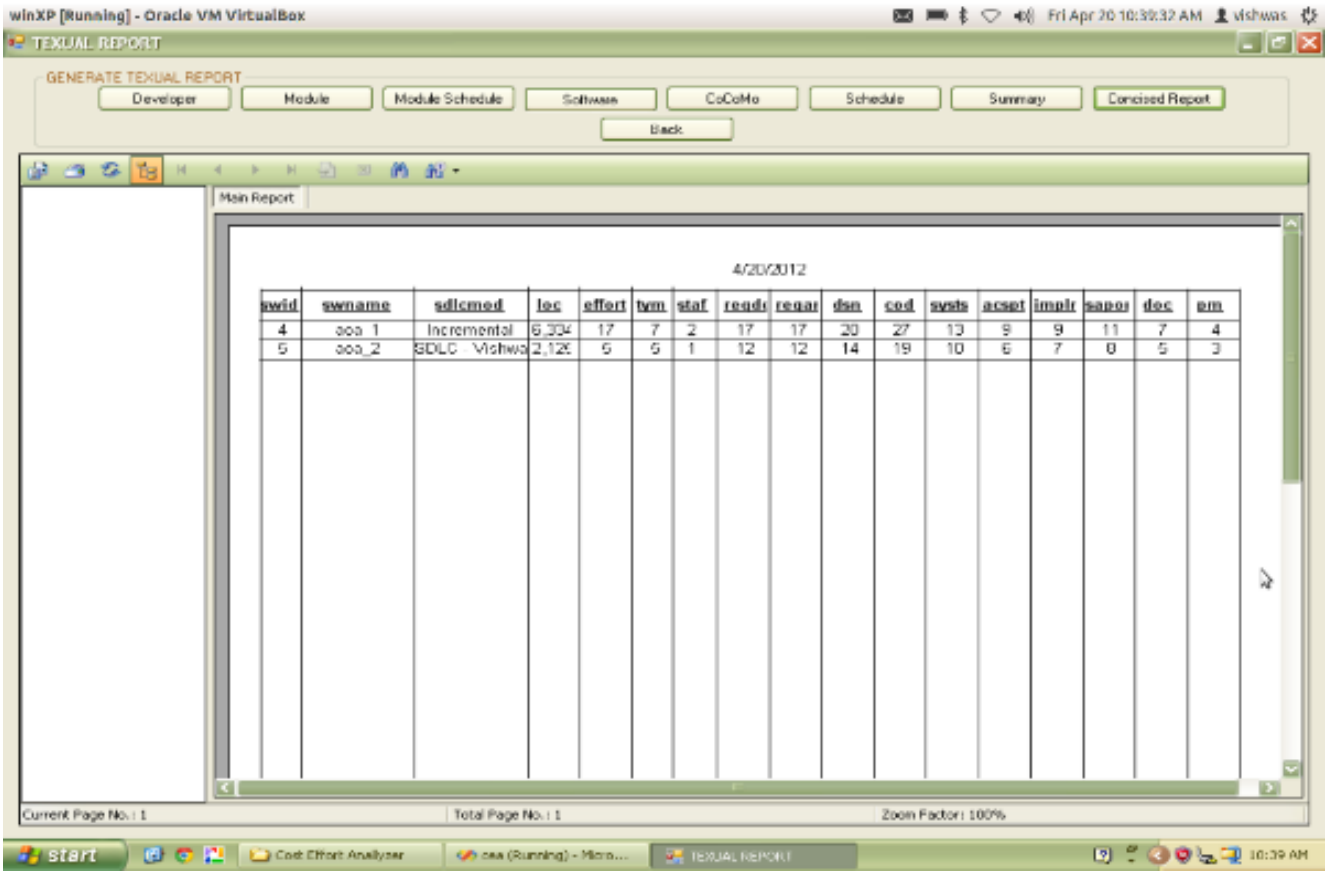

Figure 7.5: concise textual report depicting total LOC, effort, development time, staff requirement and schedule details for final released softwares

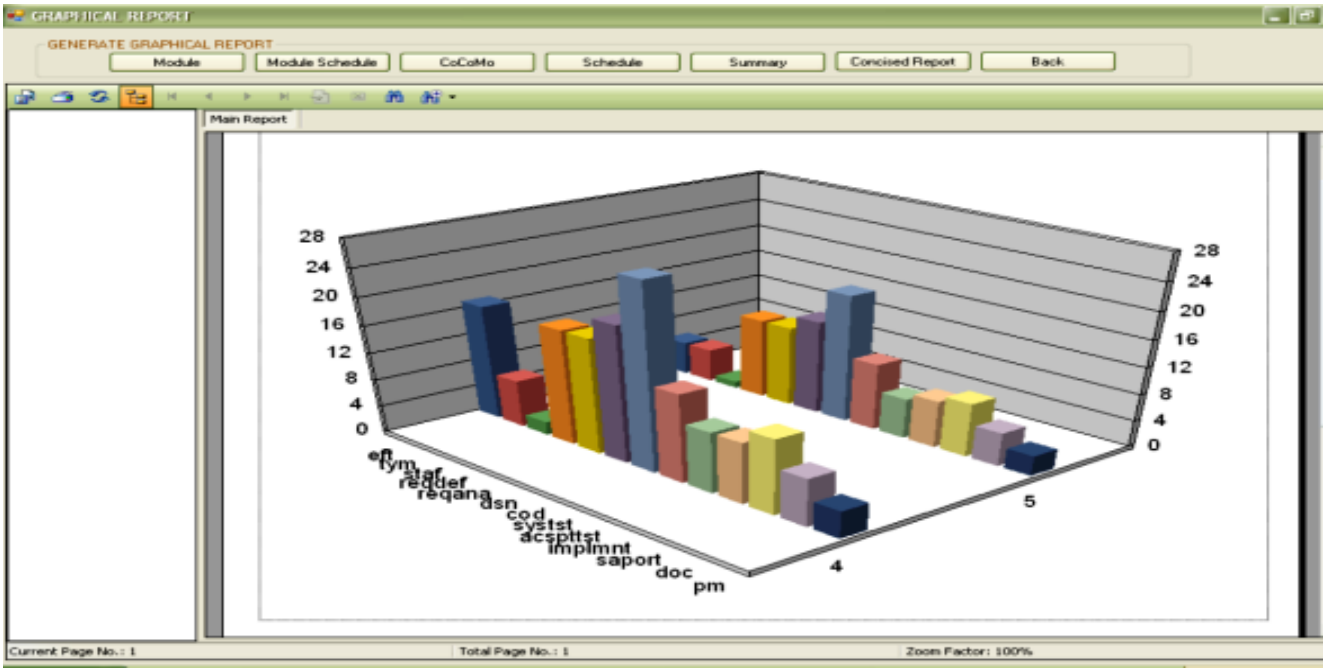

Figure 7.6: Report in form of chart depicting total LOC, effort, development time, staff requirement and schedule details for final released softwares

IX. CONCLIUSION

All the SDLC models are theoretical in nature so the SDLC "VISHWAS" model is. But these are the theoretical practices that are highly accepted and widely used in developing the software's. The IT industry employs different SDLC models for development depending upon the resources available, client for whom the software is being developed, developer capabilities, market trends, team leaders and project manager's skills etc. There is no perfect ideological situation or condition that states use of specific SDLC model for software development of specific type. The SDLC "VISHWAS" is highly suitable and recommended to be used when the client needs keeps on changing at much rapid pace and market trends gets altered quickly. Employing this model is highly recommendable for stand alone software's that are meant for specific person, company or organization.

X. FUTURE SCOPE

The above model can be extended with risk management, purchase management, incident management and configuration management. The software cea (*cost effort analyzer*) could even be developed by incorporating much advance cost estimation models like CoCoMo – II or COSYSMO.

REFERENCES

1. Software Development Life Cycle (SDLC) – the five common principles.htm
2. Roger Pressman, titled Software Engineering - a practitioner's approach
3. Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007 42 Software Project Planning(by narender sharma (istk))) The Constructive Cost Model (CoCoMo) Constructive Cost model (CoCoMo) Basic Intermediate Detailed Model proposed by B. W. Boehm's through his book Software Engineering Economics in 1981.
4. Barry Boehm. Software Engineering Economics. Englewood Cliffs, NJ:Prentice-Hall, 1981. ISBN 0-13-822122-7
5. Roger S. Pressman, Software Engineering: A Practitioner's Approach http://www.selectbs.com/analysis-and-design/what-is-the-waterfall-model
6. Roger S. Pressman, Software Engineering: A Practitioner's Approach http://en.wikipedia.org/wiki/Incremental_build_model#Incremental_Model
7. Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. Software Cost Estimation with CoCoMo II (with CD-ROM). Englewood Cliffs, NJ:Prentice-Hall, 2000. ISBN 0-13-026692-2
8. Software Release Management, 6th European Software Engineering Conference, LNCS 1301, Springer, Berlin, 1997
9. Hoek, A. van der, Wolf, A. L. (2003) Software release management for component-based software. Software—Practice & Experience. Vol. 33, Issue 1, pp. 77–98. John Wiley & Sons, Inc. New York, NY, USA.
10. Software Release Management: Proceedings of the 6th European Software Engineering Conference, LNCS 1301, Springer, Berlin, 1997(Andre van der Hoek, Richard S. Hall, Dennis Heimbigner, and Alexander L. Wolf Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, Boulder, CO 80309 USA).

AUTHORS PROFILE

Vishwas Massey did his B.E from C.I.T. Rajnandgaon, Chhattisgarh, India. Currently pursuing M.Tech in software Engineering from Rungta College of Engineering & Technology, Bhilai, India.

Prof. K. J. Satao is a Professor in Computer Science & Engineering at Rungta College of Engineering & Technology, Bhilai(C.G.). He has obtained his M.S. degree in Software Systems from BITS, Pilani(Rajasthan) in 1991. He has published more than 25 Papers in various reputed Journals, National & International Conferences. He is a Dean of the Computer Engineering & Information Technology in Chhattisgarh Swami Vivekanand Technical University, Bhilai. He is a member of the Executive Council and the Academic Council of the University. He is a member of CSI and ISTE since 2000. He has worked in various other Engineering Colleges for about 24 Years and has over 4 Years industrial experience as well. His research & development work is equivalent to Ph.D. degree in Computer Science & Engineering. His area of research includes Operating Systems, Editors & IDEs, Information System Design & Development, etc.