Appendix A: List of Publications

A. Referred Journals

1. **MANDAL A.** and PAL S. C., Achieving agility through BRIDGE process model: An approach to integrate the agile and disciplined software development. *Innovations in System and Software Engineering: A NASA Journal*, 11(1):1–7, 2015.

2. **MANDAL A.** and PAL S. C., Identifying the Reasons for Software Project Failure and Some of their Proposed Remedial through BRIDGE Process Models, *International Journal of Computer Sciences and Engineering (IJCSE)*, 3(1):118–126, 2015.

3. SARKAR S. and **MANDAL A.**, Comparison of Some Classical Edge Detection Techniques with their Suitability Analysis for Medical Images Processing, *International Journal of Computer Sciences and Engineering (IJCSE)*, 3(1):81–87, 2015.

4. SAHA D. , **MANDAL A.** and PAL S. C., User Interface Design Issues for Easy and Efficient Human Computer Interaction: An Explanatory Approach, *International Journal of Computer Sciences and Engineering*, 3(1):127–135, 2015.

5. **MANDAL A.** and PAL S. C., A Comparative Analysis of BRIDGE and Some Other Well Known Software Development Life Cycle Models, *International Journal of Computer Science and Engineering Technology*, 5(3):196–202,2014.

6. DUTTA J., **MANDAL A.** and PAL S. C., A Probabilistic Interval Division Method for Solving Nonlinear Equations, *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)*, 3(8):304–310, 2013.

7. GHOSH ROY A., and **MANDAL A.**, RDAT: A tool for DNA Analysis, *Salesian Journal of Humanities and Social Sciences: Technology and Society*, IV(1):56–61, 2013.

8. **MANDAL A.** and PAl S. C., Investigating and analysing the desired characteristics of software development lifecycle models, *International Journal of Software Engineering Research and Practices*, 2(4):9–15, 2012.

9. **MANDAL A.** and PAl S. C., Emergence of Component Based Software Engineering, *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(3):311–315, 2012.

10. **MANDAL A.**, DUTTA J. and PAL S. C., A New Efficient Technique to Construct a Minimum Spanning Tree, *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(10):93–97, 2012.

B. Conference, Seminar and Workshop Papers

11. LAHIRI S. N., DAWN S., MANDAL R. K. and **MANDAL A.**, Steganography based on Image Border Manipulation Technique (IBMT), *Proc. of the National Conference on Computational Technologies-2015 (NCCT'15)*, 231–236, 2015.

12. **MANDAL A.**, GPGPU Computing: Utilizing the GPU of a Computer as General Computational Purpose, *Proc. of Workshop on Application of Information Technology in the Modern Civilization-2013*, St. Joseph's College, Darjeeling, 15th May, 2013.

13. **MANDAL A.**, PAL S. C., Fuzzy Logic and Its Industrial Application, *Proc. of NSAMGT-2012*, Department of Mathematics, University of Kalyani, India, 2012.

14. **MANDAL A.** and PAL S. C., Role of Fuzzy Logic in Computing, *Proc. of UGC-National Seminar on Advances in Mathematics-2012*, Department of Mathematics, University of North Bengal, India, 2012.

15. **MANDAL A.**, SAHA S., SHAW S., UKIL A., An Approach for Image Based Steganography and Cryptography Technique, *Proc. of Research Scholars' Workshop on Computer Science and Application (RSWCSA-2012)*, Jointly organised by CSI, Siliguri Chapter and Salesian College, Siliguri, 2012.

16. **MANDAL A.**, Drivers for Processor Evolution: From Singlecore Towards Multicore, *Proc. of Research Scholars' Workshop on Computer Science and Application (RSWCSA-2012)*, Jointly organised by CSI, Siliguri Chapter and Salesian College, Siliguri, 2012.

17. **MANDAL A.** and MANDAL R. K., Use of e-Resources in Education and Research, *Proc. of UGC (ERO) Sponsored State Level Seminar on Promoting NMEICT-INFLIBNET e-Resources in North Bengal & Sikkim*, 25th Feb, 2011.

18. **MANDAL A.**, SRS BUILDER 1.0: An Upper Type CASE Tool for Requirement Specification, *Proc. of the* 4^{th} *National Conference,INDIACom-2010, Computing for National Development, New Delhi*, February 25–26, 2010.

19. **MANDAL A.** and PAL S. C., An empirical study and analysis of the dynamic load balancing techniques used in parallel computing systems, *Proc. of International conference on Computing and Systems (ICCS-2010) held on Nov.19–20, 2010 at Burdwan University*, 313–318, 2010.

20. **MANDAL A.** BRIDGE: A Model for Modern Software Development Process to Cater the Present Software Crisis, *Proc. of IEEE International Advance Computing Conference (IACC 2009) Patiala, India*, 1(1):494–500, 2009. Also available at IEEE Xplore DOI: 10.1109/IADCC.2009.4809259.

21. SARKAR T., DAS S. C., **MANDAL A.**, A Study of Computer-Based Simulations for Nano-Systems and their types, *Proc. of NATCOM NAMTECH-2009, Lucknow University, India*, 2009.

22. **MANDAL A.**, Operating System as Process Manager, *National Level Technical Paper Presentation Competition-2005*, SVICS-Kadi, 2005.

———— **X** ————

Appendix B: Copies of Some Publications

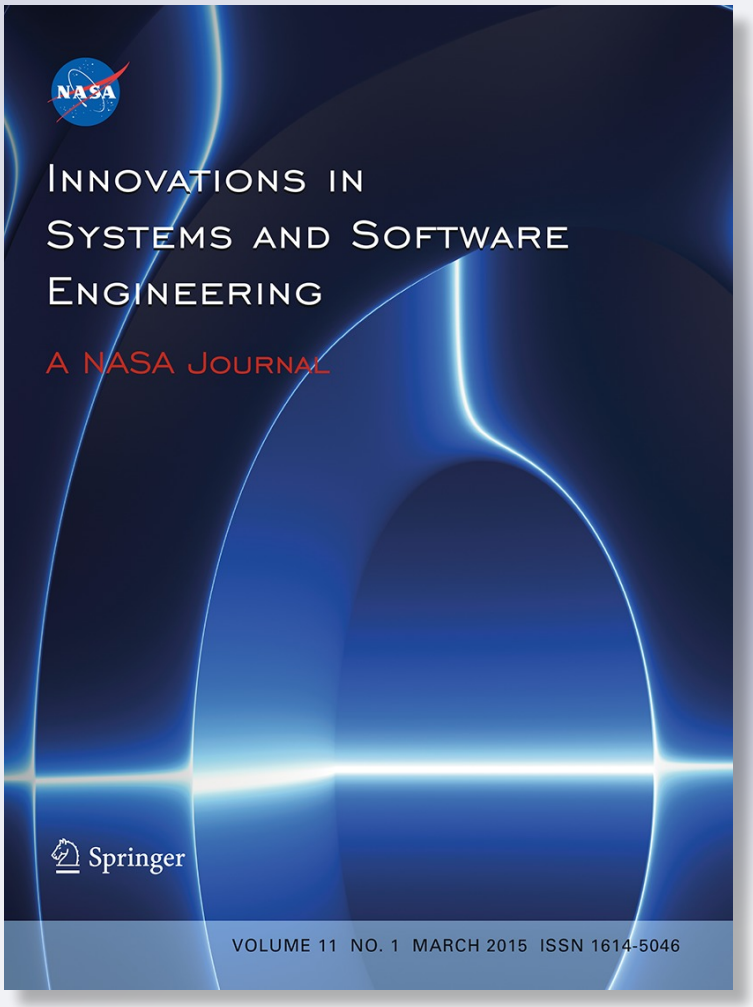
Achieving agility through BRIDGE process model: an approach to integrate the agile and disciplined software development

Ardhendu Mandal & S. C. Pal

Innovations in Systems and Software Engineering
A NASA Journal

ISSN 1614-5046
Volume 11
Number 1

Innovations Syst Softw Eng (2015)
11:1-7
DOI 10.1007/s11334-014-0239-x



Springer

Innovations Syst Softw Eng (2015) 11:1–7
DOI 10.1007/s11334-014-0239-x

ORIGINAL PAPER

Achieving agility through BRIDGE process model: an approach to integrate the agile and disciplined software development

Ardhendu Mandal · S. C. Pal

Received: 14 January 2014 / Accepted: 13 August 2014 / Published online: 23 August 2014

Abstract Despite of many criticisms, agile software development (Beck et al., Manifesto for Agile Software Development, http://agilemanifesto.org, [1]) philosophy has been proved to be quite successful to increase the project success rate up to a significant extent. The primary reasons behind the criticism of agile are the strong violation of the traditional disciplined software engineering theories, principles and practices. The goal of this research paper is to establish the fact that agility may also be achieved by following traditional process models especially with BRIDGE (Mandal, 2009 IEEE International Advance Computing Conference (IACC 2009) Patiala, India, [2]) process model. Hence, the aim of this paper is to integrate the agile and traditional disciplined software development process models and establish the compatibility between both the approaches. At the beginning, we have explored the objectives and principles of agile software development. Next, we briefly discussed the BRIDGE process model and further justified that—following BRIDGE process model, the philosophy of agile may also be achieved and conclude that BRIDGE model has both the capability of traditional software development process model as well as of the agile process.

Keywords Software engineering · Software development lifecycle · Process model · BRIDGE model · Agile software development

A. Mandal (✉) · S. C. Pal
Department of Computer Science and Application, University of North Bengal, Raja Rammohanpur, P.O. North Bengal University, Darjeeling 734013, West Bengal, India
e-mail: am.csa.nbu@gmail.com

S. C. Pal
e-mail: schpal@rediffmail.com

1 Introduction

Many software development life cycle (SDLC) process models and approaches have been introduced till date i.e., Waterfall model, Spiral model, Prototype model, Evolutionary development etc [3]. But, rarely any of these models exactly fits as-it-is for modern software development projects [4]. Hence, often instead of a single process model, most industry follows a sandwich or hybrid model i.e., mix-up of different models on demand basis [5]. Despite of this, very often the customers are unhappy with the end product and many of the projects even had to fail! Agile software development philosophy came out to be a successful approach to increase the project success rate up to significant extent while reducing the development time and cost. Despite of its success, many authors have criticized the agile approach as it often violates the basic theories, principles and practices of traditional software engineering. But the ability to meet client needs and the delivery of quality software products within estimated time is the significant benefits of agile development and is the key to its survival. Some of the additional benefits of agile process are increased productivity, expanded test coverage, improved quality, fewer defects, reduced development time and costs, delivery of better understandable and maintainable code, improved morale, better collaboration, and higher customer satisfaction as pointed out by Vijayasarathy [6]. But as said by Boehm et al. [7], neither the agile nor the traditional disciplined approach alone provide the ultimate approach. Further Hashmi et al. [8] says, there are on-going debates whether the quality of the products of the agile approaches is satisfactory. In addition, Fritzsche et al. [9] and Theunissen et al. [10] mentioned, some projects e.g., safety-critical projects require standards to be followed when developing software. Offhand, the two seem contradicting [11], [12], but several researchers agree that a software project needs

both agility and discipline [7], [13], [14]. New SDLC models are introduced at regular basis as new technology and new research results are needed to be accommodated over the time for modern project needs and suitability. This was the primary philosophy behind the introduction of BRIDGE SDLC model by Mandal [2].

2 Scope of this study

In this paper, we have just considered the theoretical aspects for the purpose of analysis and as evident. We did not consider any real development situations or data for this instance. Presently, we are running three experimental projects in our department in parallel to discover the impact of the BRIDGE process model. But as it shall take more time to complete the projects and to evaluate the successfulness, we simply skipped this for the time being. The real development situations will be disclosed in the future work when the projects will be completed and the experimental results shall be available to us.

3 From traditional disciplined SW development approach towards agile philosophy

3.1 Limitations of the traditional development models

We have many traditional SDLC models i.e., Classical Waterfall Model, Iterative Waterfall model, Spiral model, RAD model etc. in our hand by the time, but a very few are really used in practice exactly as it is. There exist several criticisms about these traditional models. According to Nandhakumar and Avison [15], traditional methods are too mechanistic to be used in detail. Truex et al. [16] pointed out that traditional SDLC models are more dogmatic and claim that traditional methods are merely unattainable ideals and hypothetical "straw men" that provide normative guidance to utopian situations. Further, Wiegers [17] noticed that, industrial software developers have become skeptical about "new" solutions that are difficult to grasp and thus remain not used. Baskerville et al. [18] claim that "to compete in the digital economy, companies must be able to develop high-quality software systems at "Internet speed"—that is, deliver new systems to customers with more value and at a faster pace than ever before". The primarily perceived limitations of the traditional development models are:

- There are too much work associated with documentation.
- They are too sequential.
- They require too much of planning activities.
- It does not show results until the end.
- It engages stakeholders too late.
- Delay in project delivery.
- Increased project cost.

For these reasons, the traditional software development approaches are rarely used in industries these days as they lack suitability for the purpose.

3.2 Agile SW development philosophy: the necessity and its origin

Agile principles evolved to address the above criticisms and primary limitations of the traditional software development. Agile software development is neither a set of tools nor a single methodology. Rather, agile is a philosophy appeared as recommendations in 2001 with an initial 17 signatories. While the publication of the "Manifesto for Agile Software Development" [1] did not start the move to agile methods, which had been going on for some time, it did signal industry acceptance of agile philosophy. Further, Kieran Conboy [19] in his paper has brilliantly derived the functional definition of agile as "the continual readiness of an information system development method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment".

Agile was a significant departure from the heavyweight document-driven traditional software development methodologies in general used at the time. Agile software development stresses rapid iterations, small and frequent releases, and evolving requirements facilitated by direct user involvement in the development process. In this way, development of agile methods could be seen as cumulative methods built on existing traditional methods where the 'good' parts are kept and the 'bad' parts are omitted or modified.

As per the recent survey report on state of agile by Versionone [20] shows that 88 % of the respondent organizations are practicing agile development, 52 % of the projects are being developed following agile.

From the Agile Manifesto [1], [21] and the definition of agility cultivated by Conboy [19], we may extract and combine the following primary feature of agile methods to be put on focus:

a. Individuals and interactions.
b. Working software delivery.
c. Customer collaboration.
d. Rapid system change incorporation i.e., responding to change.
e. Economic development.
f. Quality product development.

g. Simplicity.
h. Enhance knowledge from change incorporation.

In the next section, we give the list of principles of agile development philosophy with brief outline.

4 Principle of agile development

There were twelve basic principles of agile software development as highlighted in the Agile Manifesto [1]. The detail discussions of these principles are beyond the scope of this paper. But for the purpose of justification and comparison, we combine and highlight these principles from Agile Manifesto [1] and as extracted from Conboy's [19] definition here in brief:

1. Customer satisfaction: The highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Incorporation of rapid system change: Agile methodology welcomes changing requirements even late in development. Agile processes harness change for the customer's competitive advantage.
3. Frequent working SW delivery: Deliver working software frequently—from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Continuous cooperation of client and developer: Business people and developers must work together daily throughout the project.
5. Motivated trusted individuals: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. Continuous improvement-arrangement of face-to-face conversation: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Progress measurement: Working software is the primary measure of progress.
8. Sustainable development: Agile processes promote sustainable development. The stakeholders, developers, and users should be able to maintain a constant pace indefinitely.
9. Attention to technical excellence: Continuous attention to technical excellence and good design enhances agility.
10. Simplicity: If the design and implementation are simple, testing is easier and more effective.
11. Self-organizing teams: The best architectures, requirements, and designs emerge from self-organizing teams.
12. Internal assessment for knowledge enhancement: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
13. Quality assurance: The organization must ensure the quality of the system developed following the standard quality improvement practices.
14. Economic development: Economic development should be achieved through optimum utilization of the resources through lean system development [19].

Some authors [21], [22] have done study about scaling agile methods in regulated environments. But practicing agile in regulated environments will imply many constraints to the agile process that may be either against the philosophy of agile or it could be equivalent to a traditional development approach. As pointed out by Fitzgerald et al. [21], "Some of the essential characteristics of agile approaches appear to be incompatible with the constraints imposed by regulated environments". They further mentioned in the same paper that agile software development methods are faced with some fundamental challenges in regulated environments as a core characteristic of regulated environments is the necessity to comply with formal standards, regulations, directives and guidance. Further, as mentioned by Turk et al. [23] that agile methods and regulated environments are often seen as fundamentally incompatible. Thus, it may lead the agile process to be unlike a traditional process that is not the objective of agile always. The objective should not be to go away from the agile philosophy rather to be adhered to the Agile while achieving the advantages of disciplined approach. It is better not to tailor the agile methods to achieve the discipline necessary in regulated environments, but to optimally achieve the objectives of agile through some disciplined approach in regulated environment.

In the following section, we review the BRIDGE model as for reference and then explain how to achieve agility through this model.

5 Introduction to BRIDGE: a model for modern software development process to cater the present software crisis

The BRIDGE process model was introduced and presented by Ardhendu [2] in IEEE IACC, 2009. Although, detailed discussion of the BRIDGE model is beyond the scope of this paper, but we give the schematic diagram in Fig. 1 of the model for the sophisticated readers just for reference with its primary properties.

5.1 The primary properties of BRIDGE model

The primary properties of BRIDGE model are enlisted below [2]:

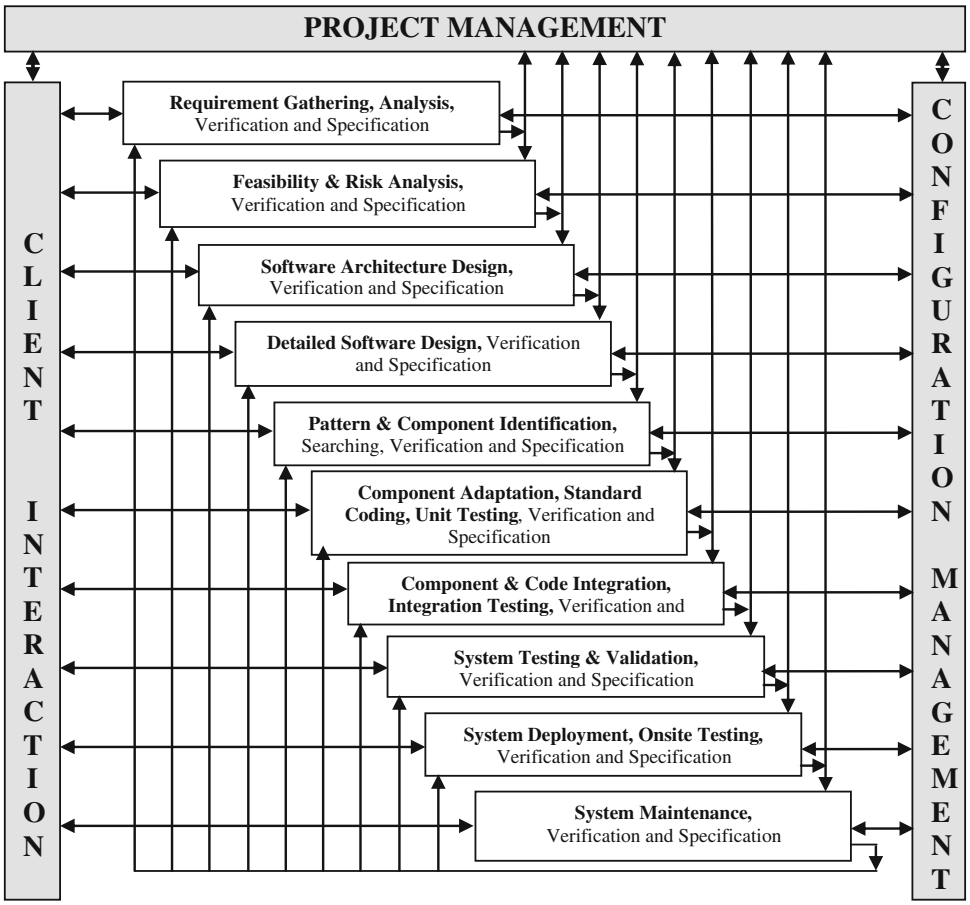


Fig. 1 The BRIDGE Process Model

a. It involves the client over the entire development life cycle activities.
b. It keeps continuous communication among the development team, project management team and client.
c. It enforces explicit verification of individual phases.
d. Supports Components-Based Software Development (CBSD).
e. It enforces on standard coding practices.
f. It considers configuration management as a separate activity.
g. It forces to specify all the phase deliverables.
h. Separate software architecture design phase.
i. Separate system deployment phase.
j. Separate on-site system testing phase.
k. It explicitly instructs to validate the system.

6 Agile development with BRIDGE model: integrating the traditional and agile philosophy

In this section, we are going to explore how the principles of agile can be achieved through BRIDGE process model. We consider all the principles of agile and discuss in brief how these are supported and can be achieved through BRIDGE model.

1. Achieving customer satisfaction: The agile principles proposed customer collaboration for increasing customer satisfaction. In BRIDGE model, this is achieved through continuous client interaction i.e., the left base pillar of the model.
2. Accommodation of requirement change: The initial system requirements may not be mature enough at the very inception of the project. As the developers and client understand the system more and more over the development process, the requirements get refined, even may get modified over the time. As the client is engaged over the entire process, as soon as new requirement is discovered, it is accommodated within the same design or necessary modification is made in the design and its subsequent phases by means of phase iteration. The iterative nature of the BRIDGE process model with the focus on component-based software development facilitates accommodation of requirement changes easily in the system. Promoting system architecture design and component assembly-based system development methodology [24], it is compara-

tively easy following BRIDGE process model to discard, add or modify system requirements.

3. Frequent working SW delivery: As BRIDGE is an iterative process, hence we may start with the initial system requirements and design the initial working software with limited capabilities and deliver to the client at earliest. As the time moves, more and more requirements can be accommodated and delivered to the client in the subsequent software versions and variants.
4. Continuous cooperation of client and developer: This is one of the best features of the BRIDGE model. The client remains as an active member of the development process from the very inception till the end of the entire process. Hence, unlike agile in this model also there exist a close continuous cooperation between the client and developers.
5. Motivated trusted individuals: Alike the client, the management unit also remains as a continuous part over the BRIDGE process model that enables the management to monitor the individual developers. Often, if needed, the management may keep on motivating the developers to give their best and may take necessary actions to make them enough trust worthy for the organization. As a result, over the time, the developers become more motivated trusted individuals for the organization. The management may investigate regularly about the need and working environment that can be allocated optimally so that the job can be done within time and budget while maintaining the quality.
6. Continuous improvement-arrangement of face-to-face conversation: We know face-to-face conversation is the best way to convey and share information. Being an integral part of the process, the management may arrange face-to-face conversation among the developers and even with the client for continuous improvement. As in BRIDGE process model, management team is associated with the development over the development period, the management team may organize such events to promote continuous improvement easily.
7. Progress measurement: In general, before starting any phase it must satisfy some phase entry criterion. As each phase of the BRIDGE model has to go through a strict verification activity before starting the immediate next phase, hence all phases have to satisfy the phase exit criteria too. The phase entry and phase exit criterion is nothing but some intermediate milestones in addition to some other desirable constraints. These milestones may be even partial working software. Through these intermediate milestones, both the development and management team may assess and measure the progress periodically. Following the way, in BRIDGE model, we may measure the progress of the project at regular interval.
8. Sustainable development: Continually evolving, growing, and changing are a natural phenomenon of any organism—none of the other organisms can survive without it. The same thing can be applied to software also, just for analogy. Very few softwares are written once, installed, and then never changed over the course of its lifetime. As per Lehman's first law [25] regarding software, a software product must change continually or become progressively less useful. New requirement will get discovered over time, some old requirements may need to be modified or discarded for the products' survival and its enhancements. Hence, the system has to be designed and developed keeping the view of anticipating the future changes in mind. Following such type of development is what called sustainable development. But unfortunately, it is a rare practice as it may increase the product cost and other development burdens. Maintaining and enhancing software to cope with newly discovered problems or new requirements can take more time than the initial development of the software. Sustainable development requires a singular focus on a form of technical excellence that regularly provides useful software to customers while keeping the cost of change minimal. Under the umbrella of effective management, the project stakeholders can maintain consistent work pace and speed promoting sustainable development following the BRIDGE model as all of the stake holders work together in this model.
9. Attention to technical excellence: In addition to design phase, the BRIDGE model has a specific software architectural design phase. Further, being the customer, an integral part of the development process, technical excellence can be monitored by both the development and project management team. Continuous attention to architectural design, low-level design and technical excellence of BRIDGE model enhances the agility.
10. Simplicity: The notion of simple is very subjective and giving practical guidelines what is simple and how to accomplish that is impossible. At the beginning, requirements seem to be quite complex, but after the analysis and as the project development progresses, they become clear. The developers should only implement features that have been agreed upon with the customers, nothing more. The art of maximizing the amount of work not done is essential. Cockburn and Glass [26] warn that simplicity should not mean neglecting design by starting programming as soon as possible. This principle most strongly supports the design of high-quality architecture and implementation. This principle further acknowledges that in programming, it is more difficult to make simple design than cumbersome solutions. Following a disciplined and systematic approach makes any process simple. The system requirements should be simple and must specify the scope of the project very specific and clear. If the design and

implementation are simple, testing becomes easier and effective. As in BRIDGE process model, all the phases are distinctly well defined, it promotes simple design and implementation of the system that makes the other subsequent activities easier, simple and more effective.

11. Self-organizing teams: By self-organizing team, we mean that the team members share a common goal and belief that their work is interdependent and collaboration is the best way to accomplish their goal. Rather than having a manager with responsibility for planning, managing and controlling the work, the team members share increasing responsibility for managing their own work and also share responsibility for problem-solving and continuous improvement of their work processes. Hence, the empowered team members' reduce their dependency on top management as they accept accountability. The team structure places ownership and controls close to the core of the work. The primary role of the project management is to build individual stakeholder a dedicated responsibility center which is easy to establish through BRIDGE process model. By developing individual responsibility centers, the team may become the self-organizing team.

Advantages of self-organizing:

a. People in a self-organized team are able to make decisions themselves and accordingly adapt to changing situations.
b. Self-organized teams do a much better job of utilizing the talents of the team because more minds are involved in any activity.
c. Self-organized teams have much more communication between team members.
d. The best way to learn is to have actual responsibility and opportunities to do new things.
e. A self-organized team is collectively aware of the upcoming work and much better able to bootstrap themselves with new work when they complete their existing task.
f. Self-organized teams spread knowledge around much better and make decisions together. That makes each team member more effective because they have much more background on the "why" of the coding assignments.
g. A command and control team member often lacks an understanding of why a decision was made because they were not involved with that decision. This may hamper their ability to follow a design or approach which restricts productivity.

As in BRIDGE model, the project management has consistent involvement with the development team, so if needed then the top management can facilitate the development team at any moment to be self-organized.

12. Internal assessment for knowledge enhancement: In association to project management, the individual stakeholders may carryout internal assessment at regular intervals. Through the internal assessment result, the progress may be measured too. Further from the internal assessment, the team may identify the various bottlenecks and upon rectifying and adjusting become more effective and plan the future activities efficiently. Being project management team with the development team in the BRIDGE process model, internal assessment becomes much easier.
13. Quality assurance: In BRIDGE model, quality ensures through implementation of multi level quality improvement methods through phase-wise verification, unit level, integration, and system testing, and system validation. Further, during maintenance, discovered errors if any may be rectified. In addition, being management team working together with the development team, the entire process and individuals remain under proper control and monitoring of the management teams. Overall, the integrated project development environment of this process model ensures the system quality to be achieved and improved.
14. Economic development: Economic development is achieved through optimal utilization of the resources and restricting misuse of resources. The optimal resource management is done through management authority with individual's care and concern. In BRIDGE, all the stakeholders work together with better coordination and concern ensuring economic system development.

7 Conclusion

Despite of the criticism towards traditional software development process, the goodness of agile process is questionable. Agile may not be the good choice for some projects always. Often, traditional process model proves to be better than agile for some types of projects. Hence, our objective should not be to criticize the traditional process models over agile, rather we must carry out research to accommodate the good attributes of agile process in traditional process models. In this paper, we have shown that the philosophy of agile may also be achieved through the BRIDGE process model which follows the principle of traditional software development too. Hence, we recommend BRIDGE process model to be practiced by industries for modern software development projects.

Acknowledgments We would like to thank the advisor of for his/her valued comments that helped us to improve the paper significantly.

References

1. Beck K, Beedle M et al (2001) Manifesto for agile software development. http://agilemanifesto.org
2. Mandal A (2009) BRIDGE: a model for modern software development process to cater the present software crisis. In: IEEE International Advance Computing Conference (IACC 2009) Patiala. IEEE Xplore, India. doi:10.1109/IADCC.2009.4809259
3. Pressman RS (2005) Software engineering: a practitioner's approach, 6th edn. Mc-Grawhil, New york
4. Cho Juyun (2009) A hybrid software development method for large-scale projects: rational unified process with scrum. J Issues Inform Syst X(2):340–348
5. Purcell JE (2007) Comparison of software development lifecycle methodologies. SANS Software Security, San Antonio
6. Vijayasarathy LEOR (2008) Agile software development: a survey of early adopters. J Inform Technol Manage XIX(2):1–8
7. Boehm B, Turner R (2003) Observations on balancing discipline and agility. In: Proceedings of the Agile Development Conference, IEEE Computer Society, Salt Lake City, pp 32–39
8. Hashmi SI, Baik J (2007) Software Quality Assurance in XP and Spiral-A Comparative Study. In: International Conference on Computational Science and its Applications, Proceedings of ICCSA-2007, IEEE, Fukuoka, pp 367
9. Fritzsche M, Keil P (2007) Agile methods and CMMI: compatibility or conflict? e-Informatica. Softw Eng J 1(1):9–26
10. Theunissen WH, Kourie DG, Watson BW (2003) Standards and agile software development. In: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology, South African Institute for Computer Scientists and Information Technologists, Johannesburg, pp 178
11. Turner R (2002) Agile development: good process or bad attitude? Lecture notes in computer science, pp 134–144
12. Turner R, Jain A (2002) Agile meets CMMI: culture clash or common cause? Lecture notes in computer science, pp. 153–165
13. Boehm B, Turner R (2003) Using risk to balance agile and plan-driven methods. IEEE Comput 36(6):57–66
14. Nawrocki J, Olek L, Jasinski M, Paliswiat B, Walter B, Pietrzak B, Godek P (2006) Balancing agility and discipline with xprince. Lecture Notes Comput Sci 3943:266
15. Nandhakumar J, Avison DE (1999) The fiction of methodological development—a field study of information systems development. Inform Technol People 12(2):175–191
16. Truex DP, Baskerville R, Travis J (2000) Amethodical systems development: the deferred meaning of systems development methods. Account Manag Inform Technol 10(1):53–79
17. Wiegers KE (1998) Read my lips: no new models. IEEE Softw 15(5):0–13
18. Baskerville R, Levine L, Pries-Heje J, Ramesh B, Slaughter S (2003) Is Internet-speed software development different? IEEE Softw 20(6):70–77
19. Conboy K (2009) Agility from first principles: reconstructing the concept of agility in information systems development. Inform Syst Res 20(3):329–354
20. VersionOne (2013) 8th Annual State of Agile Survey
21. Fitzgerald B, Stol K, O'Sullivan R, O'Brien D (2013) Scaling agile methods to regulated environments: an industry case study. In: Proceedings of 35th International Conference on Software Engineering (ICSE), San Francisco
22. Cawley O, Wang X, Richardson I (2010) "Lean/agile software development methodologies in regulated environments-state of the art". In: International Conference Lean Enterprise Software and Systems, LNBIP 65
23. Turk R France, Rumpe B (2005) Assumptions underlying agile software development processes. J Datab Manag 16:2005
24. Mandal A, Pal SC (2012) Emergence of component based software engineering. Int J Adv Res Comput Sci Softw Eng 2(3)
25. Mall R (2009) Fundamentals of software engineering 2nd edn, PHI Publication
26. Cockburn A (2001) Agile software development. Addison-Wesley, Boston

International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Special Issue-1 E-ISSN: 2347-2693

Identifying the Reasons for Software Project Failure and Some of their Proposed Remedial through BRIDGE Process Models

Ardhendu Mandal*
Dept. of Computer Science and Application,
University of North Bengal,
Dist-Darjeeling, India
am.csa.nbu@gmail.com

S. C. Pal
Dept. of Computer Science and Application,
University of North Bengal,
Dist-Darjeeling, India
schpal@rediffmail.com

***Abstract*— There are enough evidences of software project failures. Starting from economic losses to live losses is caused by many software project failures. Software project failures have significant impact on both social and economic factors. Hence, it is important to identify the different reasons for project failures. If these reasons are pre-known, actions can be taken during project development to reduce project failure risks. In this paper we identify and categorized the project failure root causes based on their different sources. Then briefly we have highlighted the primary features of the BRIDGE [1] process model and explored the ways and means how these project failure reasons may be reduced or alleviated by following the BRIDGE process model.**

Keywords- Software Engineering, Project Failure, BRIDGE Process Model, SDLC Model

I. INTRODUCTION

Software project failures are one of the primary reasons for increased cost of software product and services. There are enough evidences of project failures in past and present. Any organizations have to compensate the cost of the failure projects from the success projects. For these reason, software are still beyond the scope of small and medium scale companies causing significant impact on both social and economical factors. Apart from this, starting from economic losses to live losses is also caused by software project failures. Hence, it is important to identify the different reasons for software project failures. If these reasons are pre-known, actions can be taken during project development to reduce project failure risks.

At the beginning we have discussed about the criterion to evaluate a software project to be called successful or failed. Then, we have identified, categorized and briefly discussed different the root causes of project failures based on their source areas. Next, we have briefly highlighted the primary features of the BRIDGE [1] process model and explored the various ways and means to reduced or alleviate these project failure reasons by following the BRIDGE process model.

II. RESEARCH GOAL AND OBJECTIVES

The goal of this work is to identify the different reasons for software project failure and categorization of those reasons based on their originating sources. Further, we have tried to find out the project failure risks especially

originating from software process model and to propose their remedial strategy specially through following the BRIDGE [1] process model.

III. DEFINITION OF SUCCESSFUL AND FAILED SOFTWARE PROJECTS

The primary objective of software engineering is to develop software that agreed upon functionality and:

a. Within Time
b. Within Budget, and
c. With Good Quality

Any software development project that satisfies the above criteria is to be called successful. According to Keider [2] and Saleh [3], a project should deliver agreed upon functionality on time and within estimated budget. Successful software project maybe defined as any software project that is set to support initially-approved functionality, as well as the project comfortably satisfying the stakeholders and being accepted and largely used by the end users after deployment. Hence, Software project failure is defined as any project that is set to support the operations of an organization by exploiting the resources of information technology that fails to deliver the intended output within the originally allocated cost, time schedule [4].

IV. PROJECT FAILURE STATISTICS

To highlight the importance of this study, in this section some statistical data about the software project failure are shared. The survey statistics about software project failure and project estimate overrun carried out by Standish Group International i.e. the CHAOS Manifesto [5], in 2013 are given in Table 1 and Table 2:

Table 1: Project Performance Statistics

Year	Successful	Challenged	Failed
1994	16%	53%	31%
1996	27%	33%	40%
1998	26%	46%	28%
2000	28%	49%	23%
2002	34%	51%	15%
2004	29%	53%	18%
2006	35%	46%	19%
2008	32%	44%	24%
2010	37%	42%	21%
2012	39%	43%	18%

Table 2: Project Estimates Overrun Statistics

Year	Time Overrun	Cost Overrun	% of Features Delivered
2004	84%	56%	64%
2006	72%	47%	68%
2008	79%	54%	67%
2010	71%	46%	74%
2012	74%	59%	69%

From the statistical data presented in Table 1, it is observed that the alterative year average of project successful rate is 30.3%, project challenged by 46% and project failed by 23.4%.

From the statistical data presented above in Table 2, it is observed that the alterative year average of project time overrun rate is 76%, project cost overrun 52.5% and project feature delivery rate is 68.4%.

Rupinder Kaur and Dr. Jyotsna Sengupta in their paper [6] presented the following statistical data:

- As per the Research Report of ESSU (European Service Strategy Unit), 57% of contracts experienced cost overruns, 33% of contracts suffered major delays, 30% of contracts were terminated, and 12.5% of Strategic Service Delivery Partnerships have failed.
- As per the KPMG Survey, on average, about 70 % of all IT-related projects fail to meet their objectives.
- From the presentation on software failure by Bob Lawhorn following statistics are presented:
  - Poorly defined applications (miscommunication between business and IT) contribute to a 66% project failure rate, costing U.S. businesses at least $30 billion every year.

- 60% – 80% of project failures can be attributed directly to poor requirements gathering, analysis, and management.
- 50% are rolled back out of production
- 40% of problems are found by end users
- 25% – 40% of all spending on projects is wasted as a result of re-work.
- Up to 80% of budgets are consumed fixing self-inflicted problems (Dynamic Markets Limited 2007 Study)

Research indicates that more than 50% of all IT projects become runaways--overshooting their budgets and timetables while failing to deliver the expected outcomes [4, 7].

Johnson [4] reported that the overall project success had increased from 16% in 1994 to 28% in 2000.

That makes it very curious, but probably not surprising, that according to an article in the IEEE Spectrum, about 10% of projects are abandoned either before or after completion, because the end product will not actually resolve the original business challenge [8].

V. IDENTIFYING THE COMMON REASONS FOR SOFTWARE PROJECT FAILURE AND THEIR CATEGORIZATION

Often it is easy to identify whether a software project is successful or failed. But, it is really a tough job to identify and understand the actual reasons for project failure. For example, if the delivered system fails to meet the needs of the customer or user, the first question to ask is, "Why?":

- Was it because the development group didn't do a good job? Or
- Perhaps the requirements were not properly gathered or used? Or
- May be the people responsible for supplying the requirements were inaccurate? Or
- Was it something else?

Further, being software development a people intensive job, it is more complex to identify the exact reason to failure and to provide solutions to project failure. Usually, often there are multiple factors causing a software project to fail.

Possible areas/sources of Project Failure Reasons:

Form the above discussions it is easy to understand that there are several possible areas or sources of reasons to project failure. Some of the investigated sources of causes to software project failure are explored and listed below:

- People Sources
- Technology Sources
- Process Sources
- Organizational Sources
- Management Sources
- Business Sources
- Project Sources

Some reasons for project failure are easy to classify as belonging to one area or another, but some are harder to categorize even. So far significant effort has been made to identify and analyze the causes of software project failure discussed below [4, 6, 8, 9, 10, 11, 12, 13, and 14]**.** Now we try to identify the possible project failure reasons from the different sources as identified above.

A. Project Failure Reasons Originating from People Sources

In a software development project typically three types of stakeholders are associated:

- *Users:* Some of the project failure causes originating by these types of people may be due to:
  - Poor User Input
  - Lack of User Training
- *Client:* Some of the project failure causes originating by these types of people may be due to:
  - Conflicts

  - Politics
- *Project Development and Management Team:* Some of the project failure causes originating by these types of people may be due to:
  - Poor-Quality Work by developers
  - Poor-Quality Work by Management Personals

The project failure reasons may originate from one or more of these people sources. *Firstly,* often, the users are either unable to deliver the exact requirements to be delivered by the system or even may not be clear during initial stages of the development. As a result the project may fail due to wrong or inadequate user input. *Secondly,* often the project client and system users are different. Because of poor or misunderstanding, lack of communication gap between them, the client may convey wrong information and requirement to project development team that may lead to project failure. *Thirdly,* the project may fail because of the development team itself. These days, software development teams have become distributed in nature. The lack of communication among the development team and inefficient human resources may become the bottle neck for the project success. *Finally,* insufficient and inefficient project management team may lack to provide necessary management support to the project causing project to fail.

B. Project Failure Reasons Originating from Technology Sources

The rapid technological advancements are often good, but not always. Being software development a time intensive job, very often the technology used for the project implementation becomes obsolete before completion of the project causing the project to fail. Generally, the projects get cancelled before their completion. Further, the technology used if not chosen wrong, may be new and immature failing to perform as expected causing project failure. From technology sources SW project failure may arise due to the following reasons:

- Wrong Technology selection.
- Technology too new or didn't work as expected
- Use of immature technology
- Technology planning

C. Project Failure Reasons Originating from Process Source

Process failure is the largest and potentially the most pernicious of all sources of project failure and has been at the root of problems for decades. If the goal of a process is to produce a specific outcome, then anything that either delays or prevents the achievement of that specific outcome is a form of process failure. The process might deliver something, but if it does not deliver anticipated outcomes or does not meet expectations; the result is a failed process. This form of failure usually leads to finger pointing between development groups and users, with each claiming the other did not understand [8]. The root causes of SW project failure originating from process sources may include the following:

- **Wrong Process Selection:** There are many process models, but all have their own features and limitations. Often not all process models are suitable for any kind of projects. Thus process selection is typically challenging for any project implementation. Wrong or inappropriate process selection may lead to project fail.
- **Lack of User Involvement:** Non involvement of user and customers in the development process is one of the principle reasons that software does not fully meet customer expectations.
- **Lack of Communications:** When we think about communications failure the first thing that comes to mind is, "It's their problem," and it is usually an internal dialog. However, lack of communications with end-user or customers is rarely immediately considered, and it turns out to be one of the major problems. Further, delayed communications or communications latency is blamed as the reason for failure: "They didn't get back to me in time."
- **Unnecessary Processes:** Unnecessary processes apply to wasted or duplicated effort as well as a management or reporting structure that adds "heavy-weight" reporting and accountability to the development process.

- **Careless, sloppy, or missing software development processes:** Sloppy development process is the core value for the software engineering movement, contributed to the acceptance of object-oriented programming, helped fuel the agile movement, and more. Consider the customer at every step in the development process. While that will not guarantee the development process will be free from sloppiness, it will help focus on what is important.
- **Non-adaptability of process to Changes:** More importantly, the presence of one or more of these process failures contribute to business failure if the organization is not able to respond to changing business or market conditions. They also make it difficult to respond to customer-perceived incidents that disrupt service delivery.

D. Project Failure Reasons Originating from Organizational Sources

- New to business- lack or no prior experience
- Improper Organizational Structures in respect to project need
- Poor communication among customers, developers, and users
- Reasons Related to Human Resource
- Insufficient Resources
- Organizational culture and structure

E. Project Failure Reasons Originating from Management Sources

Additional project failure reasons may originate from project management sources. Some of the identified reasons contributing to project failure in this respect are as follows:

- Poor communication among customers, developers and users with management
- Lack of leadership and effective Management
- Poor reporting of the project's status
- Insufficient involvement of Senior management
- Insufficient staff/team Size
- Inaccurate estimates of needed resources
- Lack of proper project management and control
- Sloppy development practices
- Failure to plan
- Commitment and patterns of belief
- Poor quality management and control

F. Project Failure Reasons Originating from Business Sources

In this section we focus on different causes of failures at the business level [8] that directly affect a software development project:

- **Non adaptive to changing conditions:** One of the most obvious forms of business failure also turns out to be the primary reason that *development organizations cannot readily adapt to changing conditions*: specifically, lack of management commitment.
- **Poor selection and use of a particular tool or vendor:** Another potential source of business failure is the management requirement that dictates the use of a particular *tool* or *vendor* without considering the outcomes expected by customers.
- **Commercial pressures:** Often there is commercial pressure on the project from business sources. Time-to-market, competition in business, economic breakdown, economic competency among similar products are the different sources of commercial pressures.

G. Project Failure Reasons Originating from Project Sources

Different projects are of varying nature, types and complexity. Often, there are many intrinsic reasons to the project itself causing the project to fail! These reasons may be related to the system requirements, risks, budget, schedule etc. Some of the project related reasons originating from the project itself are identified and listed below:

- Reasons Originating from System Requirements
  i. Lack of proper understanding and poor definition of system requirements
  ii. Changing system requirements and project scope
  iii. No more need for the system to be developed
- Reasons Related to Project Risk
  i. Poor project risk identification, management and control
  ii. Late project failure warning signals
  iii. Unrealistic or unarticulated project objectives and goals
- Reasons Related to System
  i. Project's complexity
  ii. Poor system architecture and specification
  iii. Critical quality problems with software
- Reasons Related to Budget and Schedule
  i. Inaccurate/over budgeting
  ii. Hidden costs of going "Lean and Mean"
  iii. Unrealistic and over schedule estimation

The Avanade Research Report (2007) [6] disclosed that 66% of failure due to system specification, 51% due requirement understanding, and 49% due to technology selection.

Further, TCS (Tata Consultancy Services) 2007 [6] reported that 62% of organizations experienced IT projects that failed to meet schedules, 49% suffered from budget overruns, 47% had higher-than-expected maintenance costs, 41% failed to deliver the expected business value and ROI, 33% file to perform against expectations.

VI. BRIDGE PROCESS MODELS: A BRIEF HIGHLIGHTS

Although the details discussion of the BRIDGE [1] model is beyond the scope of this paper, just the schematic diagram of the BRIDGE process model is given below in *Figure 1* with its analytical results.

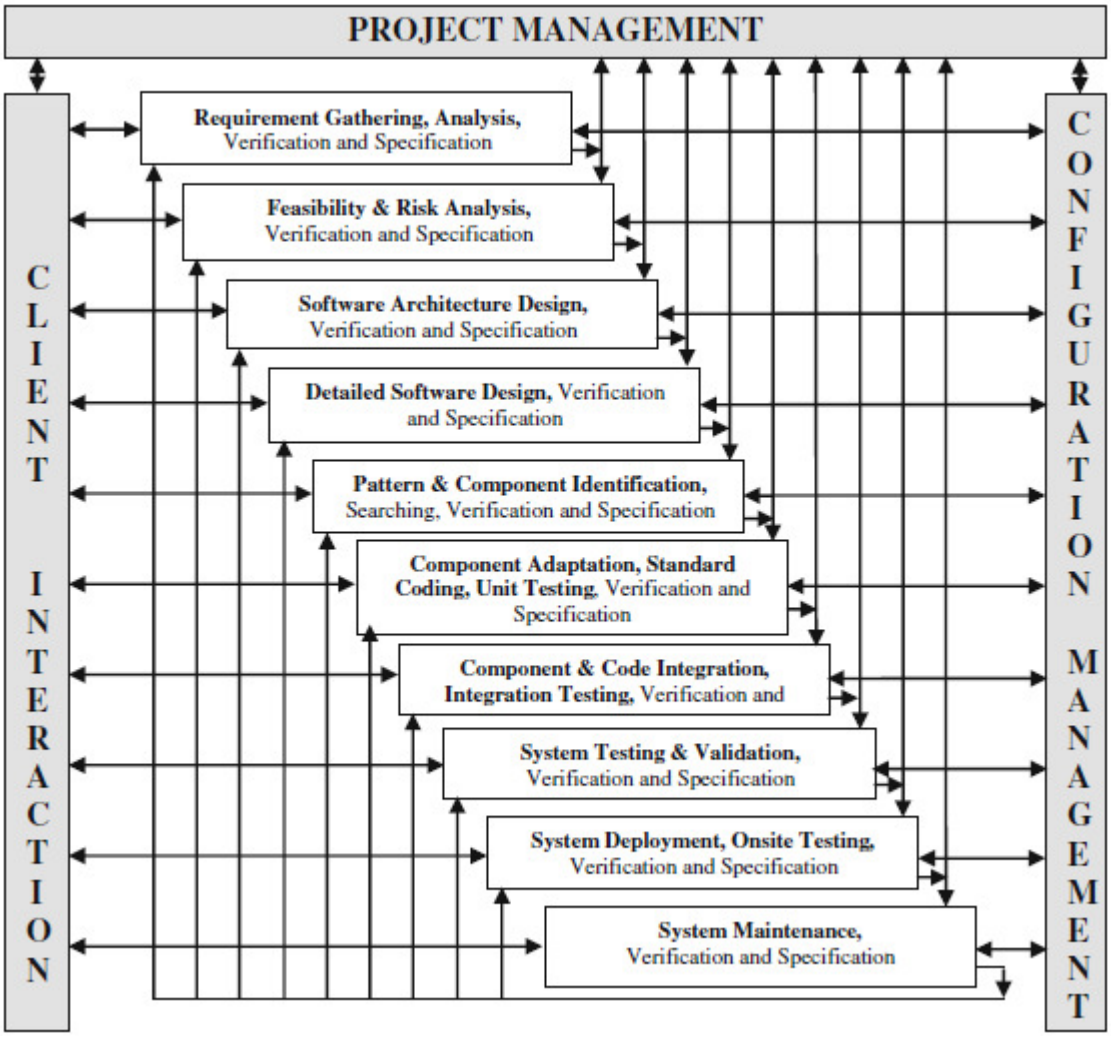


Figure 1: BRIDGE [1] Process Model

The in-depth study of the BRIDGE model discloses a lot of information that may be used to analyze the model. These are briefly discussed below [1, 15, and 16]:

- It involves the client over the entire development life cycle activities.
- It keeps continuous communication with the project management team.
- It explicit verification of individual phases.
- Separate software architecture design phase.
- Separate system deployment phase.
- Separate on-site system testing phase.
- Supports components based software development.
- It emphasizes on standard coding.
- It considers configuration management as a separate activity.
- It forces to specify all the phase deliverables.

- It explicitly instructs to validate the system.

VII. REMEDIAL TO PROJECT FAILURE RISKS TROUGH BRIDGE PROCESS MODEL

As we have discussed in the earlier sections, the project failure reasons may originate from different sources of the project i.e. people, technology, process, organizational, management business and project sources. It is not possible to address all these project failure reasons only by following any process model. However, many of these failure reasons directly or indirectly related to the software development process model. So by following any suitable process model, many of this project risk can be reduced.

In this section we discussed the remedial to some of the project failure reasons identified in the earlier sections by following BRIDGE [1] process model.

A. Remedy to Project Failure Reasons Originating from People Sources

- **Poor User Input:** In most of the process models, users are involved only during the initial phases of the development process when often the system requirements are unclear and ambiguous. Hence, the user inputs are often incorrect and poor. As the development proceeds, the requirements start getting clear. But by these times, the users are not a part of the development process. Hence, user input remains poor causing risk to project success. As in BRIDGE, the user are involved over the entire development process, the user remains the scope to provide update inputs that increases the rate of project success.
- **Lack of User Training:** Many of the organizations do think that only development and delivery of the system is the only responsibility of them. Thus they don't often take user training as serious part or their interest. But the truth is that if the users are unable to use the system easily and efficiently, the project fails irrespective of how good the developed system may be! Proper and good documentations are the key to user training but are often ignored by many organizations. In BRIDGE, special focus is given on documentations at different phases of the development process. Thus simultaneously at the end of all phases proper documents are produced that may help during the user training process and self learning.
- **Client Conflicts and Politics:** Both of these issues arise because of the lack of user involvement. The scope of these problems may be reduces only by involving the users in the development process making the user themselves to be an individual responsibility centers in the project, which is supported in BRIDGE process model.
- **Poor-Quality Work by developers and Management Personals:** There are two possible reasons for this problem to arise:
  - *Lack of Knowledge, Skill and Expertise of Developers:* In this case, the process model doesn't have any role to play; rather it is an organizational staffing and human resource related problem to be managed at organizational level.
  - *Because of Lack of Tendency to Quality-Work of Developers:* However, this reason may be handled by proper monitoring and management control efficiently by following BRIDGE as project management team is always with the development team.

B. Remedy to Project Failure Reasons Originating from Process Sources

- **Remedial to Wrong Process Selection:** The basic reasons for selecting wrong process model are unclear process objectives and goals. Further, as the different features of any process models are not distinct and ambiguous, people often selects wrong process model. In BRIDGE, the feature of the process model is very clear and unambiguous; the concerned may judge the suitability of this model for any typical project easily.
- **Remedy to Lack of User Involvement:** One of the primary features of the BRIDGE process model is the involvement of client over the entire development process that alleviates the project failure risks.
- **Lack of Communications:** This problem may also be originated from management sources. Often in no process model except BRIDGE all the stakeholders' including project management team works together. Working together by different project stakeholder in BRIDGE, the communication gap is reduced among them.

C. Remedy to Project Failure Reasons Originating from Management Sources

Remedial support to project failure causes originated from management sources are beyond the scope of any process model. For example, the quality and expertise level of the individual management and development personals don't comes under the scope of development process. Thus risks i.e. lack of leadership and effective management, poor reporting of the project's status, insufficient involvement of senior management, insufficient staff/team size, inaccurate estimates of needed resources, lack of proper project management and control, sloppy development practices, failure to plan, commitment and patterns of belief, poor quality management and control etc. depends on the quality and effective management team.

However, given an effective project management team, due to lack of direct involvement to the development process, some of the above problems may also arise, but in BRIDGE working all together under same umbrella automatically gets reduced.

D. Remedy to Project Failure Reasons Originating from Project Sources

- **Alleviating Reasons Originating from System Requirements Sources:** In BRIDGE, to alleviate reasons relate to lack of proper understanding and poor definition of system requirements, there is a dedicated phase for requirement gathering, analysis and specification that has to satisfy both the phase entry and phase exit criteria to get qualified. Further, being customer in BRIDGE always available to system analyst and developers there remains a scope to clear the doubts related to system requirements over the development process. It is also known that we may achieve the agile philosophy following BRIDGE process model [16]. Thus accommodation and adaption of changing requirement becomes easy following this process model. Moreover, as BRIDGE process model promotes Component Based Software Development (CBSD) approach [17], changing project scope becomes easy by unplugging and plugging additional software components providing services as demanded. But in case of no more need for the system to be developed, no process model can help at all, as the case with BRIDGE.
- **Alleviating Reasons Related to Project Risk:** To alleviate risks related to risk identification, management and control, and late project failure warnings signals, in BRIDGE one phase is dedicated to feasibility study and risk analysis. Further, verification activity at the end of the individual phases helps to identify and reduce these types of project failure risks.
- **Reasons Related to System Complexity:** The well known tool and technique to manage project complexity is abstraction. Using software components and CBSD approach [16], BRIDGE has the quality to handle project complexity issues. In relation to poor system architecture and specification issues, to promote CBSD, in BRIDGE there is a distinct phase for architectural design of the system apart from detailed design and at the end of the all individual phases forceful specification is mandatory.
- **Related to Software Quality Assurance:** To ensure quality system development irrespective of human related issues, BRIDGE recommends to perform verification at the end of each development phases and to perform validation and testing of the system before system deployment. Further, to ensure quality development, the organizations additionally may follow the guidelines and recommendation given by different standard bodies i.e. SEI, ISO, and Six-Sigma etc. to attain different CMM levels, ISO certifications etc.

The remedy to other project failure reasons/risks originating from other difference sources i.e. technology, organization, business are beyond the scope of capability of any process model. Further, problem related to project budget and scheduling depends heavily on the degree of expertise level of the project manager/estimator and are beyond the capability scope of any process model as the case with BRIDGE.

VIII. CONCLUSION

In this paper we have identified the different reasons contributing to project failure from there originating sources. Then we have highlighted the features of BRIDGE process model and discussed at length on how some of these project failure reasons may be reduces following BRIDGE process model. The comparative analysis of BRIDGE with some other well known process model explored the distinguished features of BRIDGE over other

[15, 18]. Thus, we conclude by recommending the BRIDGE process model to be followed for SW development projects to gain project success rate.

REFERENCES

[1] Mandal A., BRIDGE: A Model for Modern Software Development Process to Cater the Present Software Crisis, Proc. IEEE Int'l Conf. Advance Computing Conference, 2009. [Also available at IEEEXplore, DOI: 10.1109/ IADCC.2009.4809259], 494-500, 2009.

[2] Keider, S.P., Why projects fail. Datamation, 53-55, 20(12), 1974.

[3] Saleh, Y., & Alshawi, M., An alternative model for measuring the success of IS projects: the GPIS model, Journal of Enterprise Information Management, 47-63, 18(1), 2005.

[4] Walid Al-Ahmad, Khalid Al-Fagih, Khalid Khanfar, Khalid Alsamara, Saleem Abuleil, Hani Abu-Salem, A Taxonomy of an IT Project Failure: Root Causes, International Management Review, 93-106, 5(1), 2009

[5] The CHAOS Manifesto, 2013: Think Big, Act Small by The Standish Group International, 2013.

[6] Rupinder Kaur, Dr. Jyotsna Sengupta, Software Process Models and Analysis on Failure of Software Development Projects, International Journal of Scientific & Engineering Research, 1-4, 2(2), 2011

[7] Fabriek, Matthias, Elt. Al, Reasons For Success And Failure In Offshore Software Development Projects, [Available: http://is2.lse.ac.uk/asp/aspecis/20080039.pdf], [Accessed on: 05/09/2014]

[8] Alpha Software Inc, Why Software Projects Fail: A New Assessment of Risk, [Available at http://www.alphasoftware.com/documentation/200801_wpaper/low_risk_a5.pdf], [Accessed on: 05/01/2015]

[9] Lorin J. May, Major Causes of Software Project Failures. [Available at] http://www.cic.unb.br/~genaina/ES/ManMonth/SoftwareProjectFailures.pdf [Accessed on: 10.12.14]

[10] Khaled El Emam and A. Günes¸ Koru, A Replicated Survey of IT Software Project Failures, IEEE Software, 84-90, 2008.

[11] Watts S. Humphrey, Why Big Software Projects Fail: The 12 Key Questions, CROSSTALK: The Journal of Defense Software Engineering, 25-29, 18(3), March 2005

[12] Watts S. Humphrey, Five reasons why software projects fail, [Available at: http://www.computerworld.com/s/article/71209/Why_Projects_Fail?taxonomyId=11&pageNumber=2] [Accessed on: 15/12/14]

[13] By Tom Gilb. Project Failure: Some Causes and Cures, Edited MASTER paper, Version: February 29, 2004

[14] Capers Jones, Social and Technical Reasons for Software Project Failures, CROSSTALK: The Journal of Defense Software Engineering, 2-9, 19(6), June 2006

[15] Mandal A., Pal S. C., Investigating and Analysing the Desired Characteristics of Software Development Lifecycle (Sdlc) Models, International Journal Of Software Engineering Research & Practices, 9-15, 2(4), 2012.

[16] Mandal A., Pal S. C., Achieving agility through BRIDGE process model: an approach to integrate the agile and disciplined software development, Innovations in Systems and Software Engineering: A NASA Journal, 1-7, 11(1), [DOI 10.1007/s11334-014-0239-x], 2015

[17] Mandal A., Pal S. C., Emergence of Component Based Software Engineering, International Journal of Advanced Research in Computer Science and Software Engineering, 311-315, 2(3), 2012

[18] Mandal A., Pal S. C., A Comparative Analysis of BRIDGE and Some other Well Known Software Development Life Cycle Models, International Journal of Computer Science & Engineering Technology (IJCSET), 196-202, 5(3), 2014.

A Comparative Analysis of BRIDGE and Some Other Well Known Software Development Life Cycle Models

Ardhendu Mandal*

Department of Computer Science and Application, University of North Bengal, Raja Rammohunpur, P.O.- N.B.U., Dist-Darjeeling, State-West Bengal, Pin-734013, India.
am.csa.nbu@gmail.com

S. C. Pal

Department of Computer Science and Application, University of North Bengal, Raja Rammohunpur, P.O.- N.B.U., Dist-Darjeeling, State-West Bengal, Pin-734013, India.
schpal@rediffmail.com

Abstract— The existing Software Development Life Cycle Models (SDLC) models were quite successful earlier, but are rarely used in modern software development because of their limitations and non suitability for modern projects. To cater with the present software crisis, Mandal [13] proposed a SDLC model named BRIDGE for modern software development. In this paper we have outlined the BRIDGE process model. Further we performed a comparative analysis of the existing well know models and BRIDGE. Then, we discussed the results of the comparative analysis. Finally we conclude by recommending the BRIDGE process model to be the best generic process model for software development suitable for modern software development projects.

Keywords- Software Development Process Model (SDLC), BRIDGE Process Model, Comparative Analysis.

I. INTRODUCTION

The rapid development in the hardware technology has made modern processors very efficient and powerful. Hence, the expectations from the software have gone to zenith. But the complexity of the modern software are much complex as compare to those of earlier. Development cost, time and quality of the modern software are in crisis. There are several Software Development Life Cycle (SDLC) Models i.e. Classical Waterfall, Spiral, Prototype, V-Model, evolutionary model etc. All these SDLC models have several advantages as well as some limitations. A software (SW) project, irrespective of its size, goes through certain defined stages, which together, are known as the Software Development Life Cycle (SDLC). Life Cycle refers to the different phases involved starting from the project initiation to project retirement. For better understanding and implementation of the various phases of software development, different software development models have been developed and proposed so far. A few well known models are waterfall model, spiral model, evolutionary model, prototype model, V model etc. It is pre established that different SDLC models have different capabilities and limitations. Hence, selecting suitable SDLC model for any project is quite crucial as not all process models are good for any type of project. Hence, analyzing the different SDLC model is significant and helps one to select the appropriate model for a project. Recently a few more new process models are proposed with the well known traditional models to accommodate the new industrial needs.

II. SOFTWARE DEVELOPMENT APPROACH, PROCESS AND PROCESS MODEL

It is really tough to draw a sharp line between software development approaches and SDLC process models. In many literature of software engineering, these terms are used interchangeably or confusedly. So, before we begin the details discussion of the topic, let us somehow draw the boundary line between software development approaches and SDLC process models. Defining these two terms are beyond the scope of this paper. Here we just try to explain both only to establish the differences from our point of view. SW development process or simply process typically defines the set steps to be carried out during the development of the system. SW development life cycle (SDLC) is the time from the concept development to the product retirement i.e. the time of SW process. SW development life cycle (SDLC) process model typically depicts the fashions in which the SW process to be carried out i.e. which steps/phases to be done before or after another step/phase. In general all the process models do cover all distinct phases defined by SW process, but in different manner or sequence- which makes one process model differ from the other. In other words, a software development process model is an approach to the Software Development Life Cycle (SDLC) that describes the sequence of steps to be followed while developing software projects [10, 18]. We consider Agile, incremental, extreme and iterative as approach or philosophy to software development rather than as process model which can be implemented following other process models i.e. Waterfall, RAD, Spiral, Prototyping or alike.

III. DIFFERENT WELL KNOWN SDLC PROCESS MODELS

Many people have proposed different software development process models. Many are quite same in different aspects while other differs. Here we just consider some well known SDLC process models enlisted below:

Waterfall Model: It is a software development model with strictly one Iteration/phase. In this process model, development proceeds sequentially through the phases: requirements analysis, design, coding, testing, integration, and maintenance [23].

Evolutionary: Evolutionary development uses small, incremental product releases, frequent delivery to users, dynamic plans and processes. The evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle. The users provide feedback on the product for the planning stage of the next cycle and the development team responds accordingly by changing the product, plans, process etc [7, 17].

Prototype Model: It is a software development process that begins with requirements collection, followed by prototyping and user evaluation. This model facilitates to discover new or hidden requirements during the development [8].

Spiral Model: This process model proposes incremental development, using the waterfall model for each step, with more emphasis on managing risk [3].

V-Model: This is an extension of the waterfall model which emphasizes parallelism of activities of construction and verification. Here, the process steps instead of moving down in a linear way bend upwards after the coding phase resulting in the typical V shape formation.

RAD Model: It is a software development process that allows usable systems to be built in as little as 60-90 days, often with some compromises.

The details discussion of these SDLC model is beyond the scope of this paper, but just highlight the features of these models which are important for considerations. The readers may follow the references for further detail discussion of these process models [16, 21, 15]. In the following section we just briefly explain the SDLC model BRIDGE which is our prime concern.

IV. BRIDGE PROCESS MODEL IN A NUTSHELL

Although the details discussion of the BRIDGE model is beyond the scope of this paper, just the schematic diagram of the BRIDGE process model is given below in *Figure 1* with its analytical results.

The in-depth study of the BRIDGE model discloses a lot of information that may be used to analyze the model. These are briefly discussed below [13]:

- It involves the client over the entire development life cycle activities.
- It keeps continuous communication with the project management team.
- It explicit verification of individual phases.
- Separate software architecture design phase.
- Separate system deployment phase.
- Separate on-site system testing phase.
- Supports components based software development.
- It emphasizes on standard coding.
- It considers configuration management as a separate activity.
- It forces to specify all the phase deliverables.
- It explicitly instructs to validate the system.

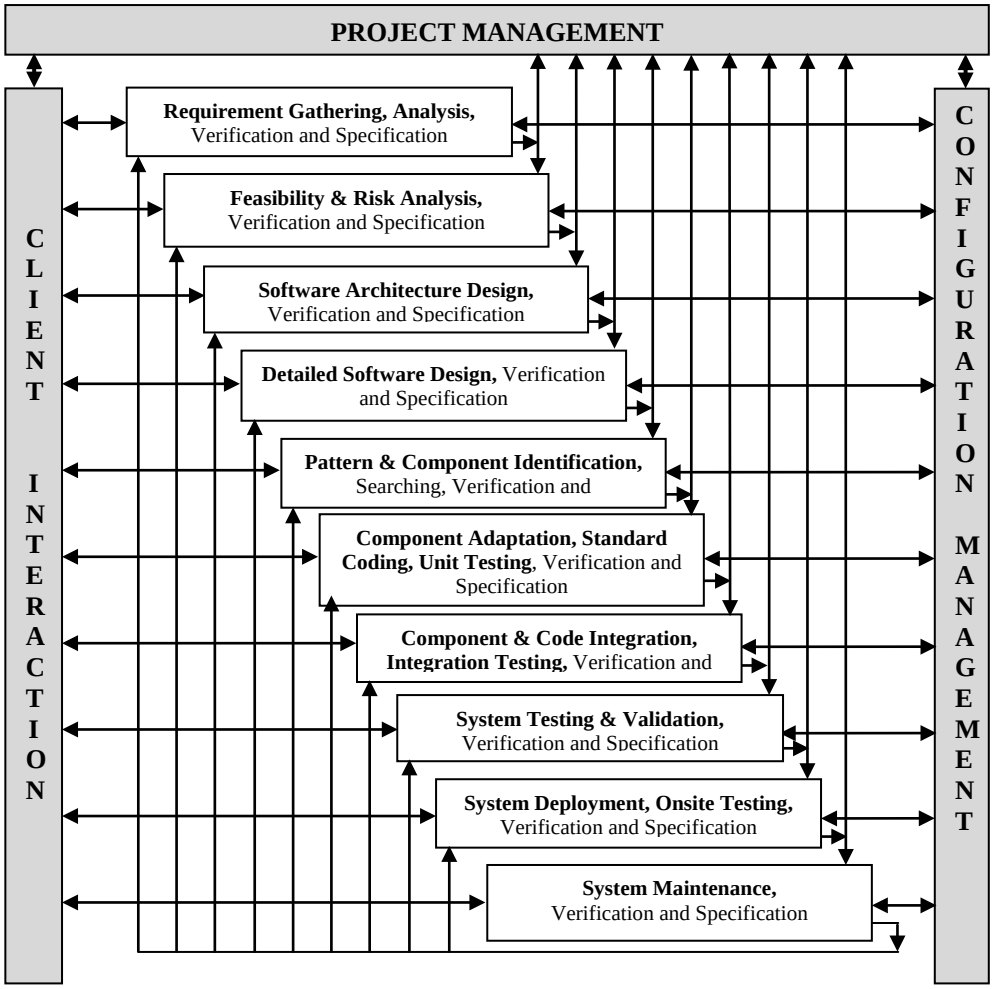


Figure 1. BRIDGE Process Model [13]

V. PARAMETER SELECTION FOR COMPARATIVE ANALYSIS

Below we enlist and discuss briefly the parameters alphabetically those we have considered for the purpose of comparative analysis:

Adaptability: This is the ability to react to operational changes as the project is developed. Change orders are easily assimilated without undue project delay and cost increases.

Budget: Budget remains one of the most significant crisis for software development projects. Some process model like Spiral and Prototyping increases the project cost as compared to others. Hence a SDLC process model has great impact on software development cost or budget.

Changes Incorporated: Change is unavoidable in software development. Managing change is a critical component of any SDLC model. Change Management and SLDC are not mutually exclusive. Change management occurs throughout the development life cycles which need to be incorporated in the system development.

Complexity of the SDLC: Different SDLC process model have varying degree of complexity. Some are easy to use and implement while others are not.

Documentation: Documentation of software development process is very important but time consuming and expensive. To reduce development time and cost, agile philosophy recommends less document which remains one of the most important critic of agile philosophy. Documentation plays vital roles in system development, implementation, maintenance and project management. But, not all process models facilitate and recommend adequate and sufficient documentation.

Expertise Required: Although some process models are better over others, but need some kind of expertise during its use and implementations in various phases at varying degrees. To avail the advantages of some process models i.e. Spiral, BRIDGE [13] and others which supports reusability- the software engineers required certain level of expertise.

Flexibility: The freedom afforded to software architects, analysts or developers to tailor the software development process according to business needs and project characteristics is a crucial factor in successful project completion. The software development organization often can benefit from introducing flexibility into their software development methodologies [20].

Guarantee of Success: This is really crucial to measure whether any process model will guarantee success or not. If so, up to what extent wills the process model guarantees the success is a big question need to be explored. As the project success depends upon many other constraints and parameters, but given the other parameters as desired, project success may vary from following one SDLC model to another.

Integrity & Security: Including security early in the system development life cycle (SDLC) will usually result in less expensive and more effective security than adding it to an operational system. To be most effective, information security must be integrated into the SDLC from its inception [9].

Maintenance: Systems are dynamic and the model offers the ability to produce a final project that is inherently designed for maintenance. This includes such items as cumulative documentation.

Management Control: Management will have the ability to redirect and if necessary redefine the project once it is begun. A key phrase is 'incremental management control', with each step under tight management control. Management control has great impact on project success.

Overlapping Phases: Each step of the project is to be completed before another is begun. Project modules are distinct and easily identifiable.

Parallel development: Parallel development support, if possible to employ may increase productivity and reduce development time while optimally utilizing the resources.

Productivity: The SDLC must ensure that the expected return on investment (ROI) for each project is well defined. The SDLC must minimize the unnecessary rework. It must be designed in such a way as to take maximum advantage of the computer assisted software engineering (CASE) tools. At the same time the SDLC must utilize the resources most effectively and efficiently to improve the productivity.

Progress Measurement: Progress measurement allows development team as well as the project management team to determine how well tasks were estimated, how well they were defined, and whether items are completed on-time and within-budget. Any SDLC process model should provide the facility to measure the progress during the system development.

Quality Control: Each module of the project can be thoroughly tested before another module is begun. Project requirements are measured against actual results. Milestones and deliverables can be used for each step of the project.

Requirements Specification: Depending on the project nature, the requirement may be identified at the very beginning of the project development or may be discovered during the development process. But, not all process model supports requirement discovery over the development process. Hence, requirement specification may be static or may be dynamic. Any SDLC process model should take into account the issue of requirement specification.

Requirements Understanding: Some process model needs the requirement must be well understood before the development process stated, while other may allow understanding the requirements over the development process. One may start with the initial understanding of the requirement and during the development the requirement understanding increases gradually.

Reusability: Reusability is one of the most significant and efficient attribute of any SDLC process model these days. Reusability helps to improve system productivity, reduce cost and system delivery on-time. The degree of reusability support may vary from one process model to another.

Risk Involvement: The risk involvement may vary from one model to another depending on the nature of requirement understanding capability support by the process model. Apart from this, there may be several other sources of risks involvement.

Risk Management: Different types of risks are implicit part of any project. Levels of risk are identifiable and assessment strategies available. Strategies are proved for over-all and unit risks.

Table: 1 Comparative Analysis

Models / Parameters	BRIDGE	Waterfall	Prototype	Evolutionary	Spiral	RAD	V-Shape
Adaptable	Excellent	Limited	Good	Good	Excellent	Limited	Limited
Budget	Low	Low	High	Low	High	Low	Low
Changes Incorporate	Easy	Impossible / Difficult	Easy	Easy	Easy	Easy	Difficult
Complexity	Medium	Simple	Moderate	Complex	Complex	Medium	Simple
Documentation	Yes	Strong	Weak	Moderate	Moderate	Poor/ Limited	Yes
Expertise Required	Medium	Low	Low	Low	High	Medium	Medium
Flexibility	Flexible	Inflexible	Highly Flexible	Highly Flexible	Flexible	High	Rigid
Guarantee of Success	High	Less	Good	Good	High	Good	High
Integrity and Security	High	Vital	Weak	Weak	High	Vital	Limited
Maintenance	Easily Maintained	Least Glamorous	Routine Maintenance	May be overlooked	Typical	Easily maintained	Lest
Management Control	Yes, Dedicated	No	No	Weak	Moderate	Weak	Weak
Overlapping Phases	May be	No	Yes	Yes	Yes	No	No
Parallel Development	Supported	No	No	Limited	Limited	No	Limited
Productivity	Highest	High	Improved	Improved	High	Improved	Improved
Progress Measurement	Measurable	Easily Monitored	Measurable	Measurable	Measurable	Measurable	Measurable
Quality Control	Very Good	Poor	Moderate	Good	Good	Adequate	Moderate
Requirements Specification	Adaptable /Dynamic	At the Beginning	Frequently Changed	Frequently Changed	At the Beginning	Time-box Release	At the Beginning
Requirements Understanding	Well Understood	Well Understood	Not Well Understood	Not Well Understood	Well Understood	Easily Understood	Easily Understood
Reusability	Excellent	Limited	Poor	Poor	Moderate	Moderate	Moderate
Risk Involvement	Low	High	Low	Moderate	Low	Little	Low
Risk Management	Highly Supporter	Not Considered	Moderate	Good	Highly Supporter	Poor	No
Simplicity	Intermediate	Simple	Simple	Intermediate	Intermediate	Very Simple	Simple
System Delivery	Early and periodic partial operational system	At the end of the system development	At the end of the system development	Early and periodic partial operational system	At the end of the system development	At the end of the system development	At the end of the system development
Time	Shortest	Short	Long	Long	Long	Short	Short
Understandability and Implementation	Moderate	Easy	Easy	Moderate	Complex	Moderate	Easy
User Involvement	Throughout Process	At the beginning	High/Up to design phases	Throughout Process	High	Throughout Process	At the beginning

Simplicity: Any model need is easy to understand and to implement. Simplicity of any process model reduces the burden of expertise and improves productivity while reduce development cost and project risk.

System Delivery: The system may be delivered either partially as individual operational module wise or as the complete system with full functionality at once.

Time: Time is actually referred to as Time Horizon because we are interested in knowing the projected completion of the project. The development time may vary from one process to another.

Understandability and Implementation: Different process model may need varying level of expertise. Simple and better understandable process model are always easy to implement.

User Involvement: Any model lends itself to strong and constant end-user involvement. This includes project design as well as interaction during all phases of project development.

VI. COMPARATIVE ANALYSIS

The comparisons among different SDLC models in respect to the features discussed above are illustrated in *Table 1* [2, 12, 11, 6, 19, 22, 5, 14, 1, 4]. From the above comparative analysis, it is established that the BRIDGE process model possesses many suitable features in comparison to the other process model.

VII. CONCLUSION

There exist several well known SDLC process models. One process model has different comparative advantages from the others in many respects. But no process model is just good for any type of project. So it is not blindly recommended to choose any process model for any project! The above comparative study shows that overall the BRIDGE process model has several competitive advantages over the other existing well known process models. As BRIDGE model has excellent adaptability, supports process tailoring and other attributes, we recommend this SDLC process model to be used for any types of software development projects.

VIII. FUTURE WORK

In near future we would like to validate the result of this theoretical comparative analysis by means of practical experimental statistical results. We are implementing several instances of one sample project following BEIDGE and different other models individually by different teams to perform practical experimental comparative analysis. During the experimental we shall refine the BRIDGE model if necessary to make this model the best alternative among the others. Further, we are working to explore the different ways to achieve the agile philosophy following BRIDGE process model.

REFERENCES

[1] Alexander L. and Davis A., Criteria for Selecting Software Process Models, presented at COMPSAC, 1991.
[2] Ali M.N. M. and Govardhan A., A Comparison Between Five Models Of Software Engineering, International Journal of Computer Science Issues, 7(5), 2010.
[3] Boehm B. W., A Spiral Model of Software Development and Enhancement, IEEE Computer, 21(5), pp. 61-72, 1988.
[4] Comer, E., Alternative Software Life Cycle Models, Proc. of International Conference on Software Engineering, 1997.
[5] Davis, A, Bersoff, E, Comer, E, A Strategy for Comparing Alternative Software Development Life Cycle Models, IEEE Transactions on Software Engineering, 14(10), pp. 1453-1461, 1988.
[6] Dholakia P. and Mankad D., The Comparative Research on Various Software Development Process Model, International Journal of Scientific and Research Publications, 3(3), 2013.
[7] Elaine L. May and Barbara A. Zimmer, The Evolutionary Development Model for Software, Hewlett-Packard Journal, 1996.
[8] Gomma H. and Scott D. B. H., Prototyping as a tool in the specification of user requirements. Proc. of Fifth Int. Conf. on Software Engineering, pp. 333-341, 1981.
[9] Grance T., Hash J. and Stevens M., Security Considerations in the Information System Development Life Cycle, NIST SPECIAL PUBLICATION 800-64 REV. 1, 2003.
[10] Guimares L. and Vilela P., Comparing Software Development Models Using CDM, Proceedings of The 6th Conference on Information Technology Education, New Jersey, pp. 339-347, 2005.
[11] Hijazi H., Khdour T. and Alarabeyyat A., A Review of Risk Management in Different Software Development Methodologies, International Journal of Computer Applications, 45(7), 2012.
[12] Malhotra S. and Malhotra S., Analysis and tabular comparison of popular SDLC models, International Journal of Advances in Computing and Information Technology, 1(3),2012
[13] Mandal A., BRIDGE: A Model for Modern Software Development Process to Cater the Present Software Crisis, Proc. IEEE Int'l Conf. Advance Computing Conference, pp. 494-500, 2009. Also available at IEEEXplore with DOI: 10.1109/ IADCC.2009.4809259
[14] Molokken-Ostvold J. and Jorgensen M., A comparison of software project overruns - flexible versus sequential development models, IEEE Transactions on Software Engineering, 31(9), pp. 754-766, 2005.
[15] Pfleeger S. L. and Atlee J. M., Software Engineering: Theory and Practice, Pearson, 2011.
[16] Pressman R. S., Software Engineering: A Practitioner's Approach, MaGrawHill Publications, 2005.
[17] Rlewallen, Software Development Life Cycle Models, 2005, http://codebeter.com.
[18] Ruparelia N., Software Development Lifecycle Models, ACM SIGSOFT Software Engineering Notes, 35(3), pp. 8-13, 2010.
[19] Sasankar B. A. and Chavan V., Survey of Software Life Cycle Models by Various Documented Standards, International Journal of Computer Science and Technology, 2(4), 2011.

[20] Sharad Chandak S., Rangarajan V., Flexibility in Software Development Methodologies: Needs and Benefits (Executive Summary), http://www.cognizant.com/InsightsWhitepapers/Flexibility-in-Software-Development-Methodologies-Needs-and-Benefits.pdf
[21] Sommerville I., Software Engineering, Pearson Education, 8th Edition, 2009.
[22] Taya S. and Gupta S., Comparative Analysis of Software Development Life Cycle Models, International Journal of Computer Science and Technology, 2(4), 2011.
[23] Walker W. Royce. Managing the development of large software systems: concepts and techniques, Proc. IEEE WESTCON, Los Angeles (August 1970) Reprinted in the Proceedings of the Ninth International Conference on Software Engineering, pp.328-338, 1987.

Investigating and Analysing the Desired Characteristics of Software Development Lifecycle (SDLC) Models

Ardhendu Mandal and S. C. Pal

Abstract—There exist several Software Development Lifecycle (SDLC) Models, but they are rarely followed by organizations for the real project implementations as they lack suitability. On the way to investigate the reasons for non suitability of these models, it is exposed that there are insufficient parameters and metric for judging the characteristics of any SDLC model. In this paper, we have investigated, identified, enlisted and analyzed the different parameters of SDLC process models. The result of this work is of great significance as these results i) may help while developing any new SDLC model ii) may even help the development team to choose the best suitable model among the alternatives for any project and iii) the outcome of this work may further be used to design and develop metrics for SDLC characterization.

Index Terms—Software Engineering, Software Development Lifecycle (SDLC), Process Model, Characterization.

1 INTRODUCTION

SEVERAL Software Development Lifecycle Models (SDLCs) are in existence [1], [2], [3]. Over the time different people have proposed different models to meet the industrial demands. Any SDLC process model should be a repeatable, clearly documented, highly-effective and must be based on the best industrial practices. However, the traditional SDLC process models provide very insightful theory and helpful best practices, but do not provide the practical details for daily application. As a result, statistics [4] shows that SDLC process models are rarely used by organizations for the purpose they are designed and developed for. Another primary reason for not using these models is due to lack of their suitability for real life projects - which led to software crisis. While investigating the reasons for unsuitability of these models, it is identified that we lack well defined characteristic parameters for any SDLC model. Without applying the process model in real project, we do not have adequate metric to analyze the suitability and goodness of such models. Davis et al [5] has proposed a strategy long back in 1988 for comparing alternative SDLCs only based on the ability to satisfy user needs and reduced life cycle cost. In this work, we have investigated, identified and analyzed the features of any SDLC process model in general which may further be used for characterizing any SDLC process model for its suitability.

2 OBJECTIVES AND GOAL

The objective of this work is to identify the different characteristics of a good software development lifecycle model. Given these characteristics, one can judge, evaluate, predict and select the best SDLC model suitable for real projects. The outcome of this work then can be used while designing and developing new process models too. Using such metric, one may evaluate a model for its suitability, applicability and predictability of success for any project. Moreover, these can be used to design the quality, suitability and predictability metric of any process model. The outcome of this research may further be used to develop new SDLC models according to the need of the industry and even may be used while designing any new process model. Hence, the goal of this work is to develop the foundations for SDLC metric.

We shall use the term software development lifecycle process model and process model over the paper interchangeably.

3 SDLC PROCESS MODELS AND OBJECTIVES

There are numerous examples of disasters that had been caused by software failures. As the computerization of the society continues, the public risks of poor quality software will become untenable unless orderly steps are taken to improve the software processes [6].

Any SDLC process model has three primary business objectives [7]:

- Ensure the delivery of high quality systems,
- Provide strong management controls over the projects, and
- Maximize the productivity of the systems development team.

Further, these objectives can be broadly categorized from the following two perspectives:

a) The Technical Perspectives

While building a system, there remain many technical activities and issues including system definition (analysis, design, coding), testing, system installation (e.g.,

Ardhendu Mandal is with the Department of Computer Science and Application, University of North Bengal, Darjeeling, West Bengal, India. E-mail: am.csa.nbu@gmail.com

S. C. Pal is with the Department of Computer Science and Application, University of North Bengal, Darjeeling, West Bengal, India. E-mail: schpal@rediffmail.com

training, data conversion), production support (e.g., problem management), defining releases, evaluating alternatives, reconciling information across phases and to a global view and defining the project's technical strategy etc. to be resolved.

b) The Management Perspectives

When we plan to develop, acquire or revise a system, we must be absolutely clear with the objectives of that system. The objectives must be stated in terms of the expected benefits that the business expects from investing in that system. The objectives should exhibit the expected return on investments. To achieve the project objectives and goal many management related issues have to addressed and resolved. The primary management activities includes setting priorities, defining objectives, project tracking and status reporting, change control, risk assessment, step-wise commitment, cost/ benefit analysis, user interaction, managing vendors, post implementation reviews, and quality assurance reviews etc.

All the above objectives irrespective of technical or managerial, has to be achieved through some SDLC process model if possible. But, unfortunately not all the available process model does address these issues efficiently.

4 Desired Characteristics of SDLC Models

In order to meet the project objectives and goal, SDLC have to satisfy many specific requirements i.e. being able to support different types of projects and systems of varying scopes, supporting both the technical and management activities, being highly usable, and providing guidance on how to execute and install it for solving real life problems and many more. In the following section we are going to identify, enlist and discuss briefly some primary characteristics that are expected from any SDLC process model. As the degree of importance of these characteristics does vary from project to project, here we just enlist these characteristics alphabetically.

Change Management

Requirement changes are often necessary, frequent and inevitable. The drivers of requirement changes may be customer demand, technical demand, competitive demand or even governmental or business policy demand. While occasional changes are essential, historical evidence demonstrates that the vast bulk of changes can be deferred and phased in at a subsequent point. To develop quality software on a predictable schedule, the requirements must be established and maintained with reasonable stability throughout the development cycle. Changes will have to be made, but they must be managed and introduced in an orderly way. Hence, change management is a critical part of any SDLC model. As requirements are changed frequently, there is a need of streamlined flexible approach to manage these requirement changes within the SDLC model. Although requirement change management and SLDC are not mutually exclusive but the change management activities occurs throughout the development process. Further, cost of adopting changes is higher after the completion of the development. Hence, the objective should be to limit the change management activities within the initial development period as much as possible. If change is not controlled, orderly testing is impossible and no quality plan can be effective.

Concurrent and Parallel Development

If high cohesion and low coupling modular system design is possible, then concurrent, distributed and parallel system development activities can be employed. This can improve productivity, timely system delivery while reducing total development cost and optimal usage of available resources.

Coordination among project stake holders

A software project is not an individuals' job, but a collective effort towards the common goal. The success of software development projects depends on carefully coordinating the effort of many individuals across the multiple stages of the development process. Coordination—long recognized as one of the fundamental problems of software engineering —has become ever more challenging. This has led to a growing body of work on coordination in software development [8]. With the rapid advancement of Information and Communication Technilogy (ICT), the location or centrer specific project evelopment barrier are being diminishing. For optimal resource utilization, often project development activities are distributed over different development centres. Further, more the people are involved in one job, more the chances of misunderstanding and communication gap. Hence, if large numbers of people are involved and scattered over different development centres, the process model must provide mechanism for better coordination among the project stakeholders.

Cost of Life Cycle Implementation

Additional costs and overheads are the primary barrier in process model implemention. For this reasons, many organizations do not implement or follow any process model. An ideal processs model implementation should be economic, easy and justifiable. It should not require additional, special application or software purchase to effectively perform the process implementation or ongoing process management. In addition, it shoud be easily automated utilizing any internal process management software tools currently being used within an organization.

Customer Involvement and Interaction

In most of the common process model, there is no direct communication among customer, development team or project management team throutout the development

process. In traditional models, management plays the vital role of bipartite body who works as the communication channel and messenger in the communication between customer and development team. As a result always there remains some communication gap and some missing or hidden information yet to convey to the development team but with the management. As a result, often proper requirements remain unspoken or hidden to the development team. Even conveyance of information might cause a loss of knowledge, as great amount of data remains with its carrier and never get handed off to others [9]. Some interviewees suggested that the lack of direct contact between the development team and the customers could encumber the process of specifying requirements for the future. In turn handoffs among functions can cause delays and increasing risks of information being misunderstood [10]. According to interviewees, level of details is varying depending on representatives between customer and developer. As a result, the developed system is frequently not satisfactory or even lead to project failure. Allowing direct communication of development team with the customer during the entire development process could eliminate project completion time and recourses consuming non value- added-action in form of handoffs, therefore waste [11]. Hence, user or customer involvement during all phases of the project development is very important for project success and must be supported by any process model.

Proper and Sufficient Documentation

Documentation has two ways of influencing the development process. Firstly, it makes the development process easier to understand. Secondly, documentation enables easy system maintenance, which can be linked to one of the principles in lean philosophy – knowledge sharing. But, unnecessary documentation can be addressed as waste. Any process model must enforce to develop the necessary document concurrently with the development pocess, while must avoid producing unnecessary documents to prevent miss utilization or waste of resources as in the case of agile development philosophy.

Early Defect Removal

In case of any error or defect, if possible, it is always better to remove or rectify them in the earliest phases of the SDLC process model. Hence, the process model must focus on identifying the errors in the same or closest phase of the SDLC process to avoid or reduce the redo-work and cost. The best way to identify errors is to perform a close and effective verification after each and every phase and to set specific predefined phase entry and phase exit criteria effectively.

Easy to Execute

Not all process models are easy to execute. Some process execution may require additional focus than the other. But often degree of easiness in execution may affect the other evaluation criteria of the process model. Always neither all easy process models are bad nor are all complex process models good. Hence, the tradeoff must be resolved depending on the suitability project demand.

Effective Management and Control

Most of the existing traditional SDLC process models don't involve management team directly with the development team. Hence, the project management team does not have direct communication with the development and associated members. The management just remains as a silent intermediate communication body. Thus, proper management observation and control is hidden in the development process. As a result, the development process lacks proper management supervision and controls. In addition, the project has to suffer from resource shortage, risk handling, coordination and many other conflicts and problems. To overcome these problems, a direct involvement of project management team with the development team is necessary and important. The software development process must be under statistical control of the project management team to produce consistent and better result through process improvment.

Focus Towards Goal

The project objectives and goal must be well defined and specific. The process model should view software development within the context of the larger system level definition, design, and development. Further, it should recognize both the potential value of opportunity and the potential impact of adverse effects, such as cost overrun, time delay or project risks according to the system and project specifications. Hence, any process model must reflect the project scope, objectives and goal consistently during the development process.

Incremental

Software development project may be divided into distinguishable cascading phases. Before starting a phase, it may require a defined set of inputs from its immediate previous phase. The incremental methodology maintains a series of such phases. However, in the design phase development is broken into a series of increments that can be implemented sequentially or in parallel. The subsequent phases do not change the requirements rather build upon them towards the project completion. The methodology continues focusing only on achieving the subset of requirements for that development increment and continues all the way through implementation. Increments can be discrete components, functionalities, or even integration activities. Hence, through the incremental methodology, quantifiable partial solutions may be given to the customer without waiting for the entire project to be completed. The subsequent partial system may be developed in parallel to the already de-

veloped and operational partial system that may be further integrated when the second incremental development is available. The incremental process increases the degree of customer satisfaction and product quality as the defect of the delivered partial system may be identified while at operation and necessary changes may be incorporated immediately and deliver with the next increment.

Iterative

In iterative process, the development begins by specifying and implementing the partial software requirements available at the moment without waiting for the complete or full software requirements specification. Further, these partial requirements can be reviewed in order to identify additional requirements and necessary modifications are made. This process is then repeated to implement the newly identified and specified requirements producing a new version of the software at each such iteration of the process model. This allows the project team to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the software in use. Iteration enhances the ability of the project management team to efficiently address the requirements of stakeholders and to complete, review, and revises phase activities until they produce satisfactory results. The product is defined as completed when it satisfies all of its requirements.

Distributed or Multi-Site Software Development

Recent advances in information technology have made Internet-based collaboration much easier. It is now possible for a software team to draw on talented developers from around the world without the need to gather them together physically. To solve the problems like team relocation and project delay, developing software at multiple sites has been considered these days. Besides the obvious advantage of being able to tap into a much larger pool of human resources, experts working together from two or more locations can actually yield better outcomes [12]. In such cases, software managers have to be able to manage these distributed teams. They need to define sharper processes, tracked, overseen and ensure that they are followed.

Quick Implementation

A predefined solution that does not require organizations to start from the very initial level would allow organizations to exponentially cut down the time required to fully complete a process implementation effort. A process blueprint solution would drastically reduce the time and cost associated with traditional process improvement by laying an effective foundation for SDLC organizations to build upon.

Phase Length and Cycle Duration

Phase length and Cycle duration should be optimal. It is well known that long cycle and phase duration is one of the primary reasons for project failure. Further, long phase and cycle duration makes the performance measurement task more difficult. If the cycle and phase duration is long, there may be problem with resource scheduling and may promote un-optimized utilization of resources which may affect the project cost, quality and schedule. Another source of risk resides in the relatively long stages, which makes it difficult to estimate time, cost and other required resources for project completion. Further, if the cycle duration is more, the delivery of incremental and partial system delivery will get delayed that may decrease customer satisfaction. Hence, the process model must be designed in such a way that the duration of each cycle and length of each phase must be small.

Predictability

Any process model should be predictable. That is, cost estimates, schedule and quality commitments would be met with reasonable consistency, and the quality of the resulting products would generally meet the users' needs [9]. As money, time, people and many other resources are involved, and at the same time quality and customer satisfaction are prime concern, before starting the project we need to predict the future outcome from it. If the outcome is not favorable, carrying out the project is nothing but waste of resources and gaining loss! In addition, the process model should ensure that we can produce desired functions with higher quality using optimal resources in lesser time in a predictable manner. Thus, the process should have a predefined level of precision to facilitate a complete, correct and predictable solution.

Process Tailoring

There may be different kind of projects and situations when no standard process is applicable. In such cases, the process model should provide the flexibility to adopt itself with the project and situational demand maintaining the integrity and consistency of the process by permitting tailoring of the standard process. Hence, tailoring is the process of adjusting the standard process to obtain a process that is suitable for the project need and situation [13]. Thus, process tailoring facility makes the process model more flexible and adaptable.

Progress Measurement

To evaluate the project performance and management control progress measurement of the project work is very important. For this purpose, milestones need to be set at regular intervals. Otherwise the project may suffer from 99% complete syndrome [1], [2]. Effective and proper project measurement is only possible when the development process is under statistical control [14].

Prototyping

Prototyping is necessary when it is very difficult to obtain exact requirements from the customer at the begin-

ning of the project. Given the prototype of the system, user keeps giving feedbacks from time to time and according to the feedbacknecessary modifications are incorporated in the proposed or tobe developed system. By doing these, the hidden, unidentified user requirements may be discovered during the initial phases of the system development process. By doing so, project failure risks may br reduces while improving the the degree of user satisfaction and system quality.

Quality Control

Lack of quality assurance during the different phases of the development process is often a potential source of risk. Validating the product is restricted to a single testing phase lately in the development process. Hence, the testing phase is the highest risky phase, since it is the last stage wherein the system is put as a subject for testing. Thus, all problems, bugs, and risks are discovered too late when the recovering from these problems requires large rework which consumes time, cost, and effort. Milestones and deliverables can be setup or specified for each step of the project. Each module of the project is thoroughly tested before the beginning of another module. Project requirements are measured against the actual results.

Reliability

Any process model must ensure development of quality reliable systems. A potential source of risk resides in the relatively long stages of any process podel, which makes it difficult to estimate, time, cost, and other resources required to complete each stage successfully. In general, if incremental model is followed, the partial working systems are delivered periodically. It increases the reliability of the process model as reliability of the system does increase gradually during each partial product delivery to the customer.

Repeatable

A SDLC process model should be repeatable i.e. the process should be repeat in case of projects which are similar in type or belongs to similar domain. A repeatable process reduces the cost of process model implementation as it is well known, learned and experienced to the development team. Hence, repeat process will reduce the project risks and cost. The quality of the system will be better as the outcomes of the phases are predictable.

Reuse

The advantages of reuse in developments are now well established. Studies show that reuse had great impact on productivity, cost, quality, time-to-market and customer satisfaction [15]. But very few process models like BDIDGE [4] are designed keeping the view of reusability in mind. Hence, as reuse has potential advantages, support of reusability is an important desired characteristic for process model in recent days.

Risk Management

Risk is commonly defined as a measure of the probability and severity of adverse effects [16]. Risks are an inherent part of any project which must be managed in advanced (if possible) or during the development process. As all projects inherit some risks, it is desired that the process model should provide adequate scope for risk management. Project risk may be related to project staffing, resources, schedules/ budgets, technical, requirements changes etc. The SDLC model must focus on the risk associated with the project continuously so that the management can take necessary control measure to prevent project failure. To manage risks, any process model must provide direct control over the project by project management. But, it has been seen that a very few model i.e. spiral and BRIDGE [4] provides such direct management support to the development process.

Scope

Client demands never ends! The more you provide, the more they demand! Hence, if the scope of the project is undefined, satisfying customer is just a dream. The process should clearly mention what is desired to be produced and the developed product should be comparable to the defined requirements. Thus, the process model should have its specific scope as well as must limit the scope of the project. Scope of the process model bounds the suitability of the process model for different types of systems and projects.

Security Assurance

Effective security is incorporated at the onset of a project. If it is included as a requirement early in the system development and/ or acquisition process, it typically results in less expensive and more cost effective security. Waiting to integrate security until later in the process usually results in interoperability issues and increased cost. Integrating security into the SDLC begins with being able to articulate the security properties desired within the system. This process is typically cyclical in refinement beginning at the top level and drilling down into what will eventually be security specifications. There are many ways to express the high-level security requirements i.e. ISO 15408 and others.

Separation of Concern in Different Phases

As the development process is divided into different phases with distinct objectives, hence the concern of different phases should be clear-cut and well separated. Otherwise, there may be chaos during progress measurement, quality and other management problems. Without separation of concerns in different phases, it will be tough to set milestones effectivel, inturn that will make the progress measurement task difficult.

Simplicity and Flexibility

Neither all simple things are better, nor are all complex things bad! The process model should be simple to un-

derstand, easy to follow and well manageable. More the process model flexible, it is easier to manage and follow for execution. As change is inevitable due to specific nature of system projects, flexibility to accommodate changes is a basic need from a process model.

Software Process Improvement and Feedback

Software Process Improvement (SPI) is an approach to designing and defining new and improved software process to achieve basic business goals and objectives i.e. increase revenue or profit, and to decrease operating costs by manipulating or changing the software process. The objectives of software process improvement (SPI) are to process produce products according to plan while simultaneously improving the organization's capability to produce better products [17]. Perfection of any process is done via constant improvements. During the project execution or at the end, the team members give feedback in the form of reports suggesting different ways to improve the development process for the next iterations or future. One of the conditions required to improve the development process is to have the input from previous cycles so that the team's opinion and experience gathered during previous phase can be dissiminated among other teams. This supports the process improvement throughout the whole organization, by adopting one of the core lean principles such as knowledge sharing, which in this case drives forward another lean principle – perfection of the process [18]. The benefits of SPI include increased customer satisfaction, productivity, quality, cost saving and cycle time reductions.

Statistical control

As Lord Kelvin said: "when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science" [9]. Statistical control means that if the work is repeated in roughly the same way, it will produce approximately the same result. Measurement is the primary principle behind the statistical control. A SDLC model should be under statistical control of project management team. Such a process model will produce the desired results within the anticipated limits of cost, schedule and quality. If the process is not under statistical control, no progress is possible until it is [14].

Suitability to Projects

Not all process models are suitable for every type of projects. Some process models are suitable for large projects, while some may be better for small projects. But, there are some flexible process models which are suitable for different types of projects. Any process model should not be suitable to only a specific type of project or project scope. The model should be designed in such a way that it should support different types of project with their varying scope through process tailoring. Hence, suitability to projects may be considered as an evaluation criteria for a process model.

Support to the Modern Tools and Technologies

Over the time, significant developments are made in the field of new techniques and methodologies. Those are to be incorporated, accommodated and supported in the process models to make it a sustainable for modern software development. If a process model fails to accommodate these new technologies, they gradually become obsolete and useless. For example, during last few years there has been significant development in the field of CASE tools for project management, configuration management, software design, modeling and many more. It is a proven fact that usage of CASE tools increases the product quality and reduces the total project development cost and time to market. Hence, the process model should be designed to support and utilize the CASE tools.

Usability

The usability requirement addresses the various ways in which the SDLC will be used by the team members easily, efficiently while at use. Usability is a composite property of a process model. It is a composition of five attributes i.e. i) Learning ability, ii) Efficiency, iii) User retention over time, iv) Error rate, and v) Satisfaction [19]. Any process model should incorporate these usability attributes. Many usability process has been proposed by several people i.e. ACUDUC (Approach Centered on Usability and Driven by use cases) by Seffah et al [20] , Usability Engineering Process Model (UEPM) by Granollers [21] and Xavier Ferre [19] has proposed an integrated model of usability and different SDLC activities to integrate usability especially in different SDLC models. Adoption Centric Usability Engineering (ACUE) facilitates the adoption of usability engineering methods for software engineering practitioners and thereby improves their integration into existing software development methodologies and practices [22].

5 CONCLUSION

An important initial step in addressing software problems is to treat the entire development process as a performable, controllable, measurable and improved process as a sequence of tasks that will produce the desired result. Any fully effective software process must consider the interrelationships of all the required tasks, tools and methods, skills, training and motivation of the people involved. In general, any process model must bear the properties that are investigated, identified and specified in this paper. At the same time, an ideal SDLC process implementation should be quicker, cost-effective and easy to implement and follow. It should also be stakeholder and project team-friendly. Finally, the ideal

process model should address the top challenges experienced by project managers. Although the degree of importance of these individual characteristics may vary from project to project and depends on situational demand, but we recommend that all the above discussed characteristics should be beared by any process model to suit for the modern real projects.

6 Future Work Plan

In this paper we have identified and enlisted the desired characteristics from any process model irrespective of their degree of importance. In near future, we shall investigate the significance and importance of these individual characteristics and prioratise them accordingly. Further, using such priority list of process model characteristics, we shall develop a process metrics depending on that one may choose the suitable process model for any project which shall provide optimal solution and shall address other common isses related to process models.

REFERENCES

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, MaGrawHill Publications, 2005.

[2] I. Sommerville, *Software Engineering*, Pearson, 2009.

[3] S. L. Pfleeger and J. M. Atlee, *Software Engineering: Theory and Practice*, Pearson, 2011.

[4] A. Mandal, "BRIDGE: A Model for Modern Software Development Process to Cater the Present Software Crisis", *Proc. IEEE Int'l Conf. Advance Computing Conference*, pp. 494-500, 2009. Also available at IEEEXplore with DOI: 10.1109/IADCC.2009.4809259

[5] A. M. Davis, E. H. Bersoff and E. R. Comer, "Strategy of Comparing Alternative Software Development Life Cycle Models", *IEEE Trans. on Software Eng.*, vol-14, no-10, 1988.

[6] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*, Addison-Wesley, 2003.

[7] Systems Development Life Cycle: Objectives and Requirements, Bender RBT Inc. 17 Cardinale Lane, Queensbury, NY 12804, 518-743-8755

[8] Marcelo Cataldo, James D. Herbsleb; "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 39, NO. 3, MARCH 2013

[9] W. S. Humphrey, Characterizing the Software Process: A Maturity Framework, Technical Report, CMU/ SEI-87-TR-11, ESD-TR-87-112, Software Engineering Institute, 1987.

[10] M. Walton, Strategies for Lean Product Development, Massachusetts Institute of Technology, 1999.

[11] Larman and Vodde, *Scaling Lean and Agile Development: Successful Large, Multisite & Offshore Products with Large-Scale Scrum*, Ch.22, Addison-Wesley, 2008.

[12] KEITH C.C. CHAN , LAWRENCE M.L. CHUNG , Integrating Process and Project Management for Multi-Site Software Development, Annals of Software Engineering 14, 115–143, 2002

[13] M. P. Ginsberg and L. H. Quinn. Process Tailoring and the software Capability Maturity Model. Technical Report, Software Engineering Institute, CMU/ SEI-94-TR-024.

[14] W. Edwards Demming, Quality, Productivity, and Competitive Position, Cambridge, MA: Massachusetts Institute of Technology Center for Advanced Engineering Study, 1982.

[15] Mandal A. and Pal S. C., "Emergence of Component Based Software Engineering", Int'l Journal of Advanced Research in Computer Science and Software Engineeringc, Volume 2, Issue 3, March 2012.

[16] Lowrance, William W., Of Acceptable Risk: Science and the Determination of Safety. Los Altos, Ca: William Kaufmann, 1976.

[17] W. S. Humphrey, "Managing the software process", Reading, Addison-Wesley, MA, 1989.

[18] A. Antanovich, A. Sheyko, B. Katumba, Bottlenecks in the Development Life Cycle of a Feature: A Case Study Conducted at Ericsson AB, Report No. 2010:012, ISSN: 1651-4769, University of Gothenburg.

[19] Xavier Ferré and Natalia Juristo, Helmut Windl, Larry Constantine, "Usability Basics for Software Developers". IEEE Software, January/ February 2001, pg 22-29

[20] A. Seffah, R. Djouab and H. Antunes, "Comparing and Reconciling Usability-Centered and Use Case-Driven Requirements Engineering Processes" 0-7695-0969-W IEEE 2001

[21] Toni Granollers; "User Centered Design Process Model. Integration of Usability Engineering and Software Engineering" Proceedings of INTERACT, 2003

[22] Eduard Metzker, Ahmed Seffah; "Adoption of Usability Engineering Methods: A Measurement-Based Strategy"

Ardhendu Mandal, Assistant Professor, Department of Computer Science and Application, University of North Bengal, India. He has more than 5 Yrs of teaching and research experience in the field of Computer Science and Application with research focus in the field of Software Engineering especially in the area of software development lifecycle process models and its relavent aspects. His research interest also includes High Performance Computing and Bioinformatics.

S. C. Pal, Professor, Department of Computer Science and Application, University of North Bengal, India. He has more than 16 Yrs of teaching and research experience in the field of Mathematics and Compute Science with special research focus in the field of Computational Fracture Mechanics. Recently he is also working in the field of High Performance Computing and Software Engineering.

Volume 2, Issue 3, March 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Emergence of Component Based Software Engineering

Ardhendu Mandal*
Department of Computer Science and Application
University of North Bengal
Pin-734013, India
am.csa.nbu@gmail.com

S. C. Pal
Department of Computer Science and Application
University of North Bengal
Pin-734013, India

Abstract-- **It was noticed that, most software systems are not new but are variants of systems that had been already developed. Hence, a new systems may be developed partially if not completely, from the pre-existing systems by reusing it. This brings the idea of reusability and gave the birth of a noble concept of Component Based Software Development, beyond object oriented development paradigm. Component Based Software Development aims to construct complex software systems by means of integrating reusable software components. This approach promises to alleviate the software crisis at great extents. The objective of this paper is to gain attention towards this new component based software development paradigm and to highlight the benefits and impact of the approach for making it a successful software development approach to the concerned community and industry.**

Keywords— **Software Engineering, Software Components, Component Based Software Engineering, Component Interface**
Abbreviations-- **SE-Software Engineering, CBSE-Component Based Software Engineering, CBSD- Component Based Software Development.**

I. INTRODUCTION

In early days, software engineering approach was ad hoc. Around 1970s, introduction of structured programming" gave a formal shift in software engineering from the ad hoc to a systematic approach. Then around 1980s, introduction of object oriented programming with some advancement explores new areas in software engineering. In recent dates, with the introduction of Component Based Software Development (CBSD), the industry is moving in a new direction. The basic insight is that most software systems are not new. Rather, they are variants of systems that have already been built. This insight can be leveraged to improve the quality and productivity of the software production process [1]. These day's software systems are more complex as compared to those of early. These complex, high quality software systems are built efficiently using component based approach in a shorter time. Component based systems are easier to assemble and therefore less costly to build than developing such systems from scratch. The importance of component based development lies in its efficiency. In addition, CBSE encourages the use of predictable architectural patterns and standard software infrastructure, thereby leading to a higher result. In the remaining part of this paper the term *"component"* and *"software components" will be used interchangeably.*

II. THE JOURNEY OF THE COMPONENT BASED DEVELOPMENT ERA

In 1968, Douglas McIlroy's [2] first share the idea of Component Based Software Development (CBSD) at the NATO conference on software engineering in Garmisch, Germany in his paper titled "Mass Produced Software Components". He discussed the idea that, software may be componentized i.e. built from pre-developed software components. His subsequent inclusion of pipes and filters into the Unix operating system was the first implementation of an infrastructure for this idea. Later, Brad Cox set out to create an infrastructure and market for these components by inventing the Objective-C programming language. IBM led the path with their System Object Model (SOM) in the early 1990s. Some claim that Microsoft paved the way for actual deployment of component software with OLE and COM. As of 2010 many successful software component models do exist.

III. UNDERSTANDING SOFTWARE COMPONENTS

In early days, the principles of software engineering have been focused in developing software system from the very scratch development for individual software. It means that, for all the functionalities to be supported by a system have been designed and coded individually for the proposed systems under development. Brown [3] posits that it would be infeasible for developers and organizations to consider constructing each new information system from scratch.

Instead, information systems would need to be developed with reused practices, software components and products that have been tested and proven to be effective and efficient in order to remain in business and gain competitive advantage [3, 4, 5]. Upon long observation it was found that- there are certain functionalities those are common in many systems. Hence, if these common functionalities can be developed independently - may be reused in different systems without redevelopment from scratch. Later, these can be integrated to any system as part whenever it is suitable! At the same time, it will reduce the development effort of such systems as there is no need to develop the same common parts again and again for different systems. This originates the concepts of Software Components. Although, a software components is a small parts of a system, but often a large system as a whole may be seen as a software component as well. It is also possible that, the system consists of components is a component itself. In all cases, the components are required to be reusable components after all.

Historically, "component" in software is a rough synonym for "module" or "unit" or "routine". A generally accepted view of a software component is that, it is a *software unit with provided services and required services from others too* (Fig. 1). The *provided services* are operations performed by the component whereas, the *required services* are the services needed by the component to provide target services. The one or more *interface* of a component consists of the specifications of its provided and required services [6].

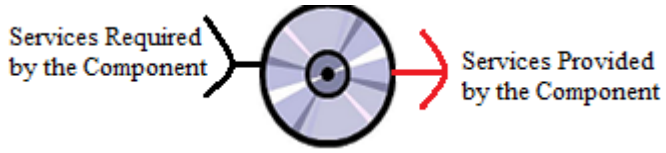


Fig. 1 Service Scheme of Software Component

As component is simply a data capsule, information hiding becomes the core construction principle underlying components. Clemens Szyperski [7] suggests shifting the focus away from code source. He defines a software component as executable, with a black-box interface that allows it to be deployed by those who did not develop it. It is important for a software component to be easily combined and composed with other software components. This is because a software component will only achieve its usefulness when it is used in collaboration with other software components [8].

According to Herzum and Sims [9], the term 'component' is used in many different ways by practitioners in the industry. However, some rather broad and general yet useful definitions are as follows:

"An independently deliverable piece of functionality providing access to its services through interfaces".

-Brown [10]

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

- Clemens Szyperski [11]

A component denotes a self-contained entity (a.k.a. black box) that exports functionality to its environment and may also import functionality from its environment using well-defined and open interfaces. In this context, an interface defines the syntax and semantics of the functionality it comprises (i.e., it defines a contract between the environment and the component). Components may support their integration into the surrounding environment by providing mechanisms such as introspection or configuration functionality.

- Michael Stal [12]

Hence, we may define a software component as: ***An independently deployable and compositional software element having specific functionality that conforms to a software architecture.***

IV. USING SOFTWARE COMPONENTS: COMPONENT BASED SYSTEM DEVELOPMENT

The *usage* of a component in a software system includes using it to replace an out of date component to upgrade the system or a failed component to repair the system, adding it to the system to extend the system services, or composing it into the system while the system itself is still being built. Some researchers insist on a component being reusable during dynamic reconfiguration [13]. CBSD advocates developing software systems by selecting reliable, reusable and robust software components and assembling them within appropriate software architectures.

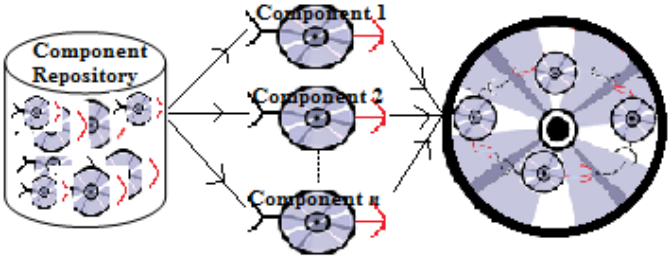


Fig. 2 Component Based System Development

With the help of middleware technologies- a set of specifications or rules in the form of functions, which when incorporated into the code allows the software to be integrated with software developed using other platforms/languages [14]. Example of such middleware technologies are COM, DCOM, CORBA, JavaBeans, EJB etc. Component Based Software Engineering (CBSE) is a process that aims to design and develop software systems using *existing, reusable* and *adaptable* software components as opposed to programming them. Hence, *CBSE shifts the emphasis from programming to composing software systems".*

V. OBJECTIVES OF COMPONENT BASED SOFTWARE DEVELOPMENT: ALLEVIATE SOFTWARE CRISIS

The goal of component-based development is to build and maintain software systems by using existing software components [15, 16, 17, 18, 19, 20]. As said by Dr. Randall W. Jensen [21], "High customer demand, reduced software development budgets, and a competitive software market drive the need for reusable software". When correctly applied and implemented, developing software systems using Commercial Off The Shelf (COTS) software components promises benefits like increase productivity, shorten time-to-market, improve software quality, reduce maintenance cost, allow for inter-application interoperability, decreased level of risks, leverage technical skills and knowledge, and improve system functionality [22, 23, 24]. As, software crisis may be loosely defined as the set problems associated with the software development process [25] i.e. quality, development cost and time-to-market of software, we may conclude that the indirect objective of CBSD are to alleviate software crisis. In addition, the particular objectives of software components are to:

a. **To have a useful 'replaceable property'** i.e. easy to assemble and easy to disassemble.

b. **Increase Reusability:** Develop once and reuse several instances of the same over the period.

c. **Facilitating System Change Management and System Maintenance: T**he plug and play feature of a component allows easy component composition and inclusion in the information systems.

d. **Enhancing Development Flexibility: C**omponents are an independent software element that can be designed and developed independently enhancing the development flexibility.

e. **Reduced System Development Time: R**eusing pre-developed existing components instead of new fresh development will reduce total development time.

f. **Reduced System Development Cost: R**educed development time will result in significant reduction in total development cost.

g. **Improve Software Quality:** Ideally, a component is pre-tested for errors and quality parameters. Hence, using such pre-tested, high quality software components improves the quality of complete software systems.

h. **Reducing Project Risk:** From management perspective, if an asset's costs can be optimized through a large number of uses, it would then be possible for the management to expend more effort and allocate more budgets to improve the quality of software components. This in turn reduces the level of risk faced by the development effort and will undeniably improve the likelihood of success [26].

i. **Improve interoperability: W**hen systems are developed using reused components, they are expected to be more **interoperable** as they rely on common mechanisms to implement most of their functions [27]

j. **Increase System Learning for User:** Dialogs and interfaces used by these systems would be similar and would improve the learning curve of users who utilize several different systems built using the same components [26, 27].

k. **Ease and Efficient System Debugging:** As the components are previously tested, if any error occurs, must be during the integration. Hence, the domain and range for debugging is minimized and localized. This facilitates the debugging process quite easy and efficient.

VI. IMPACT COMPONENT BASED SOFTWARE DEVELOPMENT: AN QUANTITATIVE ANALYSIS

The CBSD approach includes improvements in: quality, throughput, performance, reliability and interoperability; it also reduces development, documentation, maintenance and staff training time and cost [23]. In this approach, due to inherent functional independence software is assembled from components can be autonomously deployed and, the productivity and performance of the development team can be improved [9]. The impact of software reuse in system development is highlighted below:

a) Impact on Productivity

Although percentage productivity improvement reports are notoriously difficult to interpret, it appears that 30-50% reuse can result in productivity improvements in the 25–40% range. According to Lim [22], Hewlett-Packard software projects reported productivity increases from 6% to 40% with the incorporation of CBSD. Further, Pitney Bowes in the USA which has been reusing components since 1996 documented tremendous savings in labour as the company is now able to achieve 500 human-weeks of development progress in only 200 human-weeks by using existing components and by purchasing others from component markets [28].

b) Impact on Quality

In a study conducted at Hewlett Packard, Lim [22] reports that the defect rate for reused code is 0.9 defects per KLOC, while the rate for newly developed software is 4.1 defects per KLOC. Henry and Faller [27] reported that, for an application that was composed of 68% reused code, the defect rate was 2.0 defects per KLOC—a 51% improvement from the expected rate, had the application been developed without reuse. They further report a 35% improvement in quality by component reuse in system development. They again suggested that, the quality of information systems developed using this approach will also have fewer bugs and defects if compared with newly built-from-scratch systems.

c) Impact on Time-to-Market

The **STG** division reports that the same development effort using the reusable work product required only 21 calendar months compared to an estimated 36 calendar months had the reusable work product not been used, a reduction of 42% [22].

d) Impact on Cost

Apart from productivity gains, component reuse allows organizations to reduce the critical path in the delivery of

software systems, reducing the time-to-market and begin to accrue profits earlier. Study shows that there has been reduction in product cost up to 75-84% as a result of reuse [29].

VII. INDUSTRIAL PRACTICES ON SOFTWARE COMPONENTS

"The use of commercial off-the-shelf (COTS) products as elements of larger systems is becoming increasingly commonplace. Shrinking budgets, accelerating rates of COTS enhancement, and expanding system requirements are all driving this process. The shift from custom development to COTS-based systems is occurring in both new development and maintenance activities. If done properly, this shift can help establish a sustainable modernization practice."

- SEI COTS-Based Systems Initiative [30]

Because the potential impact of reuse and CBSE on the software industry is enormous, a number of major companies and industry consortia have proposed standards for component software. In recent years, component technologies have been well developed, such as Enterprise Java Beans (EJB) of Sun [31], CORBA Component Model (CCM) of the OMG [32], and Component Object Model (COM) of Microsoft [33].

A. *SUN JavaBeans Components*

The JavaBean [33] component system is a portable, platform independent CBSE infrastructure developed using the Java programming language. The JavaBean system extends the Java applets to accommodate the more sophisticated software components required for component-based development. The *Bean Development Kit* (BDK) encompasses a set of tools to facilitate the CBSD approach.

B. *OMG/CORBA*

The Object Management Group has published common *object request broker architecture* (OMG/CORBA) [32]. An object request broker (ORB) provides a variety of services that enable reusable components to communicate with other components, regardless of their location within a system. When components are built using the OMG/CORBA standard, integration of those components without modification in a system is assured if an *Interface Definition Language* (IDL) is created for every component.

C. *Microsoft COM*

Microsoft has developed a Component Object Model (COM) [35] that provides a specification for using components produced by various vendors within a single application running under the Windows operating system. COM encompasses two elements: *COM interfaces* that are implemented as COM objects and a set of *mechanisms for registering and passing messages* between COM interfaces. From application point of view, "the focus is not on how implemented, only on the fact that the object has an interface that it registers with the system, and that it uses the component system to communicate with other COM objects [34]".

VIII. CONCLUSIONS

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. Developing software using Commercial Off The Shelf reusable component is known as Component Based Software Development (CBSD). Informally, application of Software Engineering principles and practices in CBSD is known as Component Based Software Engineering (CBSE). CBSD qualifies, adapts, and integrates software components for reuse in a new system. In addition, often engineers need to develop additional components that are based on the custom requirements of a new system that are unavailable from component library. CBSD offers inherent benefits in software quality, developer productivity, and overall system cost, but yet many roadblocks remain to be overcome before the CBSD is widely used throughout the industry. We may conclude that, component based development is the future development process to cater the present software crisis. Industries must follow this development practices, built and enlarge their component library for future reuse. At the same time industries must train their developers to encourage and practice the component based software development process and need to set up appropriate facility centres for supporting CBSD process. Despite lots of potentialities, there are still lots of issues that are to be explored for being the CBSD successful. Hence, researchers have to take the responsibilities on their shoulder to resolve these issues and make CBSD a successful software development approach.

REFERENCES

[1] William B. Frakes and Kyo Kang, Software Reuse Research: Status and Future, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 31, NO. 7, JULY 2005

[2] M. D. McIlroy, Mass Produced Software Components , NATO Software Engineering Conference Report, Garmisch, Germany, October, 1968, pp. 79-85.

[3] Brown, A. W., 'Preface: Foundations for component-based software engineering', in *Component-based Software Engineering - Selected Papers from the Software Engineering Institute*, A. W. Brown (ed), 1996, pp. vii - x.

[4] Butt, J., 'Reuse Rather Than Rebuild', *eWeek*, 18(44), 2001, p. 37.

[5] Szyperski, C., *Component Software - Beyond Object-oriented Programming*, ACM Press/Addison Wesley, USA, 1998.

[6] Kung-Kiu Lau and Zheng Wang, *Software Component Models*, IEEE Transactions on Software Engineering, Vol. 33, NO. 10, October 2007.

[7] Clemens Szyperski, *Component Software*, Addison-Wesley, 2nd edition, 2002.

[8] Councill. B. & Heinernan, G. T., 'Definition of a Software Component and its Elements', in Component-based Software Engineering - Putting the Pieces Together, B Councill & G. T. Heineman (eds), Addison Wesley, USA, 2001.

[9] Herzum, P. & Sims, O., Business Component Factory- A Comprehensive Overview of Component-based Development for Enterprise, John Wiley, New York, 2000.

[10] Brown, A. W., Large-Scale, *Component-Based Development*, Prentice-Hall, USA, 2000.

[11] *WCOP'96 Summary in ECOOP'96 Workshop Reader*, Dpunkt Verlag, 1997. ISBN 3-920993-67-5.
[12] Anton Deimel, Juergen Henn, et al., *What characterizes a (software) component? Software - Concepts & Tools*, Vol. 19, No. 1. (1 June 1998), pp. 49-56.
[13] M.R.V. Chaudron and E. de Jong. *Components are from Mars.* In Proc. 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, Lecture Notes In Computer Science; Vol. 1800, pages 727-733, 2000.
[14] Sparling M.: "Is there a Component Market", www.cbd_hq.com/articles/2000
[15] G. Gossler and J. Sifakis. *Composition for component-based modeling*, Science of Computer Programming, 55(1-3), 2005.
[16] N. Medvidovic and R.N. Taylor. *A classification and comparison framework for software architecture description languages.* IEEE Transactions on Software Engineering, 26(1):70.93, 2000.
[17] R. Roshandel, B. Schmerl, N. Medvidovic, D. Garlan, and D. Zhang. *Understanding tradeoffs among different architectural modeling approaches.* In Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA04). IEEE Computer Society, 2004.
[18] J.-G. Schneider and O. Nierstrasz, *Components, scripts and glue.* In L. Barroca, J. Hall, and P. Hall, editors, Software Architectures Advances and Applications, pages 13-25. Springer, 1999.
[19] D. Hybertson. *A uniform component modeling space*. Informatica, 25:475.482, 2001.
[20] Szyperski, C., *Component Software - Beyond Object-oriented Programming*, ACM Press/Addison Wesley, USA, 1998.
[21] Dr. Randall W. Jensen, An Economic Analysis of Software Reuse, CROSSTALK The Journal of Defense Software Engineering December 2004
[22] Lim W.C., *Effect of reuse on quality, productivity and economics*, IEEE Software Vol 11, No 5, Sep 1994, pp23-30
[23] Pinto, P. E., *Promoting Software Reuse in a Computer Setting* [Online], Available: http://www-2.cs.cmu.edu/afs/cs/usr/ppinto/www/ reuse.html, [Accessed: 20 August 2002].
[24] Patrizio, A., 'The New Developer Portals - Buying, selling, and building components on the web speeds companies' time to market', Information Week, August, p. 81, 2000.
[25] Ardhendu Mandal, *BRIDGE: A Model for Modern Software Development Process to Cater the Present Software Crisis*, 2009 IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6–7March 2009
[26] Lim, W. C., *What is Software Reuse?* [Online], Available: http://www.flashline.com/content/lim/what_reuse.jsp, [Accessed: 20 August 2002].
[27] Henry E., Faller B., *Large-scale industrial reuse to reduce cost and cycle time*, IEEE Software, Vol 12, No 5, Sep 1995, pp47-53
[28] Scannell, E., *Web Services Reignite Component Reuse* [Online], Available: http://www.itworld.com/AppDev/4162/IWD010409hnreuse/pfindex.html, [Accessed: 15 August 2002]
[29] Kuljit Kaur, Parminder Kaur, Jaspreet Bedi, and Hardeep Singh, Towards a Suitable and Systematic Approach for Component Based Software Development, World Academy of Science, Engineering and Technology 27 2007
[30] Robert C. Seacord, Daniel Plakosh, Grace A. Lewis, Modernizing legacy systems: Software Technologies, Engineering Processes and Business Practices
[31] Sun Microsystems. Enterprise JavaBeans(TM) specification 2.0. *Sun Developer Network Enterprise JavaBeans Technology Official Website*. Sun Microsystems, Inc.: Santa Clara CA, 2005; 572 pp. Http://Java.sun.com/products/ejb [10 September 2004]
[32] Object Management Group. CORBA component model (CCM) 3.0. *Object Management Group Working Group Specification*. Object Management Group, Inc.: Needham MA 2002; 434 pp. Http://www.omg.org/cgibin/apps/doc?formal/02-06-65.pdf [10 September 2004]
[33] Rogerson D. *Inside COM*. Microsoft Press: Redmond WA, 1997; 376pp.
[34] Carlos Canal and Antonio Cansado, *Component Reconfiguration in Presence of Mismatch*, Informatica 35 (2011), Pages 29–37

Proceedings of the 4th National Conference; INDIACom-2010
Computing For Nation Development, February 25 – 26, 2010
Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi

SRS BUILDER 1.0: An Upper Type CASE Tool For Requirement Specification

Ardhendu Mandal
Department of Computer Science and Applications, University of North Bengal (N.B.U.), INDIA
E-Mail:am.csa.nbu@gmail.com

ABSTRACT
Software (SW) development is a labor intensive activity. Modern software projects generally have to deal with producing and managing large and complex software products. Developing such software has become an extremely challenging job not only because of inherent complexity, but also mainly for economic constraints unlike time, quality, maintainability concerns. Hence, developing modern software within the budget still remains as one of the main software crisis. The most significant way to reduce the software development cost is to use the Computer-Aided Software Engineering (CASE) tools over the entire Software Development Life Cycle (SDLC) process as substitute to expensive human labor cost. We think that automation of software development methods is a valuable support for the software engineers in coping with this complexity and for improving quality too. This paper demonstrates the newly developed CASE tools name "SRS Builder 1.0" for software requirement specification developed at our university laboratory, University of North Bengal, India. This paper discusses our new developed product with its functionalities and usages. We believe the tool has the potential to play an important role in the software development process.

KEYWORDS: CASE tool, software development, SDLC, SRS, SE, FHD.

1. INTRODUCTION

Although, hardware costs are decreasing drastically, still the computers are not used extensively by the business organization because of the huge software cost. In recent years, Computer-Aided Software Engineering tools have emerged as one of the most important innovations in software development to manage the complexity of software development projects reducing its product cost. Using CASE tools over their SDLC process may reduce the developing cost significantly.

Although, almost all develop countries do use certain CASE tools, but its extensive use is still a dream because of their product cost. Although, big software giants do use their own developed or commercially available CASE tools in development software, but their quality, applicability, cost, availability remains as big question. The huge cost of such commercially available CASE tools made these outreach of the small software development companies. In this study, we are going to demonstrate newly developed requirement specification tool name SRS BUILDER version 1.0.

The rest of the paper is organized as follows: We started by defining software engineering, Computer Aided Software Engineering and CASE tools. Then, we have laid down the different types of CASE tools and advantage of using CASE tools with its limitation. In the later section we have discussed about SRS BUILDER 1.0 with its sample design.

2. COMPUTER AIDED SOFTWARE ENGINEERING (CSAE) and CASE TOOLS

In the following we are going to define some terminologies used in software engineering.

- **Software Engineering (SE):** Before defining CASE, defining software engineering is justifiable. Although, the age of *Software Engineering* is quite old, but prior to its born, people do used to develop software. But because of some problems (discussion of these problems are beyond the scope of paper) faced later with those systems, software engineering emerged as a new subject.

The IEEE defines ***software engineering*** as [6]:

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
- The study of approaches as in (i).

More significantly, software engineering may be defined as the systematic approach to develop **quality** software **within both budget and time** constraints.

- **Computer-Aided Software Engineering (CASE):** CASE is an acronym that stands for Computer-Aided Software Engineering. Roughly, this is all about using computers at different phases of the SDLC process during the development and maintenance of software to assistance the development team. CASE provides the software engineer with the ability to automate manual activities and to improve engineering associated to software development. Basically, *it is all about using software to develop software.*
- **CASE tools:** CASE tools are the tools that permit collaborative software development and maintenance. These tools are concerned with automated tools that aid in the definition and implementation of software systems.

Formal definition of CASE:

- "Individual tools to aid the software developer or project manager during one or more phases of software development (or maintenance)."
- "A combination of software tools and structured development methodologies".

- **Types of CASE tools:** Depending on the activities performed, CASE tools are primarily divided in to three categories. They are as follows:
  - **Upper CASE tools:** Primarily focuses on the System

 ISSN 0973-7529 ISBN 978-81-904526-9-4

analysis and design phase of the SDLC.

- **Lower CASE tools:** Focuses on system implementation phase of SDLC.
- **Integrated CASE tools:** It helps in providing linkages between the lower and Upper CASE tools.

3. ADVANTAGES OF USING CASE TOOLS

With the growing importance of CASE tools, more steps in the SDLC are being automated. However, a complete automated software facility for all steps is still a dream. Presently available CASE tools cover only a certain modules of the general CASE facility.

The benefits of using CASE tools are as follows:

- Increasing Productivity.
- Product Quality improvement.
- Development Cost Reduction.
- Effort Reduction: Different studies carried out to measure the impact of CASE put the effort reduction about 30-40% [4].
- Reduction in Development Time.
- Reduce the drudgery and working style in a software engineer's work.
- *Create good quality documentation. Our, proposed tool basically focuses on this particular aspect of computer aided software engineering.*
- Create maintainable system.
- Providing a uniform platform for software/system developers to present information and knowledge compactly for ease of communication (Banker & Kauffman, 1991; Church & Matthews, 1995; Orlikowski, 1989).

4. USAGE OF CASE TOOLS: A RESEARCH REPORT

CASE (Computer-Aided Software Engineering) tools are supposed to increase productivity, improve the software

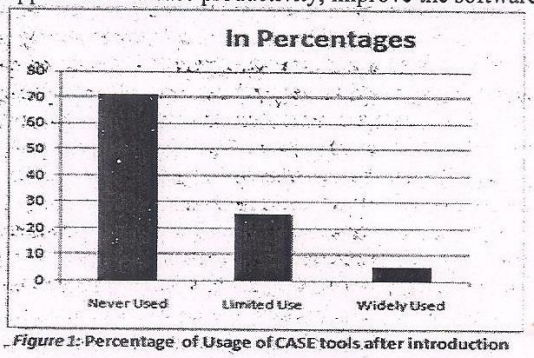


Figure 1: Percentage of Usage of CASE tools after introduction

product quality and make Information Systems development a more enjoyable task [2]. However, they have been failing to deliver the benefits they promise [1]. The results of the research carried out by Kemerer, C. F. [3] regarding the usage of CASE tools after introduction are shown in the *Figure 1.*

This results about CASE tool usage raise a significant question, "Why the percentage of widely used case tools are too low (5% only)?" The answer to the question is the limitations with the CASE tools discussed in the next section.

5. LIMITATION OF CASE TOOLS

CASE tools are still in limited use because of the following limitations:

- CASE tools don't support automatic development of functionally relevant system.
- It force system analyst to follow a prescribed methodology.
- It may change the system analysis and design process.
- Poorly supported expected functionality from them.
- Huge cost of the CASE tools.
- Lack of Concern in CASE tool usage.
- Bad quality of the CASE tools.
- CASE tools are complex because they offer a large array of options and support for software development activities.
- Cheap Labour cost: In developing countries like India and other underdeveloped countries, cheaply available human resources remain as one of the biggest reasons for not using the costly CASE tools.

6. MOTIVATION FOR UNDERTAKING THE RESEARCH PROJECT

In addition to usage in industry by practitioners, CASE tools are employed by educators to teach students software development skills. Evaluating and selecting a CASE tool for a specific systems development course is quite difficult.

While teaching the paper software engineering at university (NBU), it was found that the students are very much confused about CASE tools. The significant reasons for the same are basically widely available complex poor quality CASE tools which are quite expensive to purchase although. Some modules are good in some if not in all. The students do not feel interesting to use them. Then we under took the project to develop a new customised CASE tool for requirement specification supporting IEEE specification as per the students need that will help the student to improve their skill and have a clear vision about it. We start with the development of CASE tool to specify the system requirements to generate the System Requirement Specification (SRS) document. The outcome of our attempt is the so named, "***SRS BUILDER 1.0***"- the CASE tools to generate the SRS document. Moreover, we are planning to distribute the newly developed CASE tool to the educational institutions on demand almost free of cost with a nominal transportation cost for the sake of student community.

7. SRS BUILDER 1.0: THE NEW REQUIREMENT SPECIFICATION TOOL

As from the SLC models, we know that after the requirements are gathered by the system analyst, the analysed and finalised requirements need to be specify in the SRS document. The SRS document is then provided to the software development team for next phase of the development.

- ***SDLC Model Followed:*** We have followed the **BRIDGE** software development process model [5] to develop the *SRS BUILDER 1.0.*

 ISSN 0973-7529 ISBN 978-81-904526-9-4

- ***Product Quality Features:***
  - Support to IEEE specification for SRS writing format along with the flexible other customized specifications as per your organizational need.
  - Good Graphical User Interface
  - Easy to use
  - Easy Installation
  - Incorporated User Documentation
  - Easily Available
  - Compatible
  - Minimal System Requirements for Installation
  - Validated
- ***System Specification:***
  Front End: Visual Basic 6.0
  Back End: MySQL
  Operating System: Windows XP, Windows Vista.
  Memory Requirement: 15 KB

8. FUNCTION HIERARCHY DIAGRAM (FHD) OF SRS BUILDER 1.0

The Function Hierarchy Diagram (FHD) of the SRS BUILDER 1.0 is shown below in *Figure 2.*

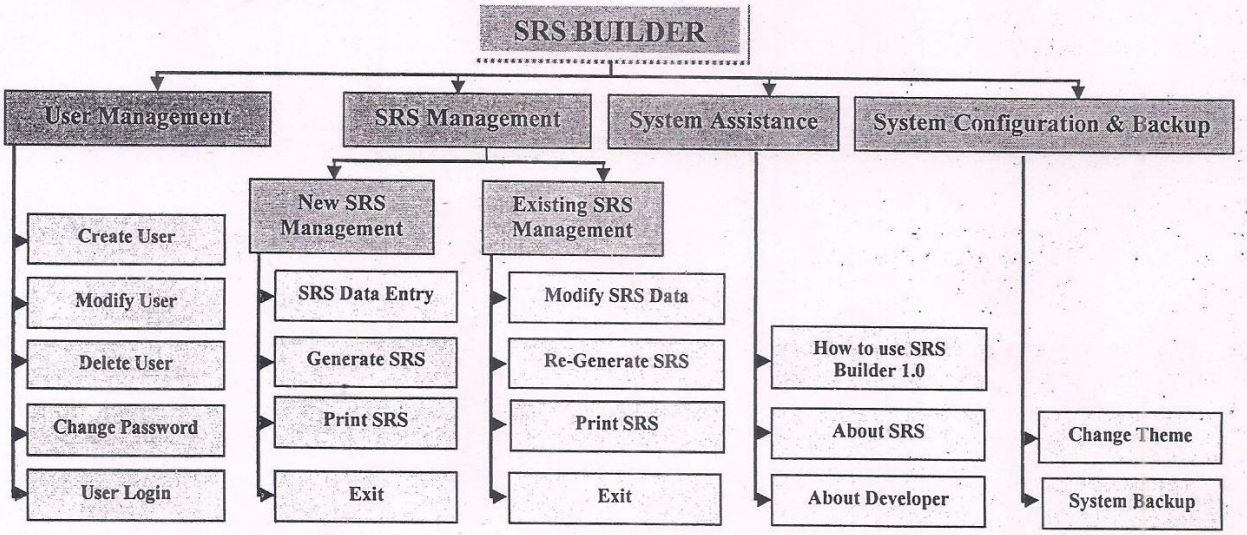


Figure 2: *Function Hierarchy Diagram (FHD) of SRS Builder 1.0*

9. SAMPLE SRS ORGANIZATION GENERATED BY SRS BUILDER 1.0

A typical SRS generated by the SRS BUILDER 1.0 for the ATM system is shown below in *Figure 3*. This is not a complete and correct SRS for the intended system, but a sample used only to show the SRS organization generated by the tool. The font size and spacing has been changed to accommodate in the paper keeping the context organization unchanged.

Project Title: SRS ATM

Project Id: 1

SOFTWARE REQUIREMENT SPECIFICATION

Introduction

Purpose:

This document describes the software require……

Scope:

The software supports a computerized banking …….

Definition:

 ISSN 0973-7529 ISBN 978-81-904526-9-4

• Account:

A single account at a bank against which transaction……..

Intended Audience:

The intended audience of this SRS consists of:

• Software designers….

Reference: NA

Overview: NA

Document Conventions: NA

Overall Description

Product Perspective:

An automated teller machine (ATM) is a……. customer is identified by inserting a plastic ATM card ……..

Product Function:

1. Get Balance Information…..

User Characteristics:

Open to all authorized users..... Customers are simply members of the public with no special training……

Operating Environment: Ability to read the ATM card……

General Constarints: NA

User Documentation: NA

Assumptions Dependencies: Hardware never fails ……

Specific Requirements

N.A……

External Interface Requirements

User Interface:

The customer user interface should be intuitive, such that 99.9% of all new ATM users are able……..

Hardware Interface:

• Ability to read the ATM card.....

Software Interface:

• State Bank……

Communication Interface:

• List of Communicational interface equirements ……

Functional Requirements:

• List of functional requirements ……

Behavioural Requirements:

 ISSN 0973-7529 ISBN 978-81-904526-9-4

• List of behavioural requirements .....

Other Non-functional Requirements

Performance Requirements:

• It must be able to perform in adverse conditions like high/low temperature etc. .....

Safety Requirements:

• Must be safe kept in physical aspects, say in a cabin .....

Security Requirements:

• Users accessibility is censured in all the ways .....

Software Quality: NA

Other Requirements: NA

SYSTEM REQUIREMENTS SPECIFICATION for ATM Withdrawal

Submitted by:

Program Manager/Functional Project Officer	Date
Coordination:	
Director, Applications Architecture	Date
Director, Engineering	Date
Test Director	Date
Approved by:	
Functional Manager	Date

Figure 3: A typical SRS organization generated by SRS BUILDER 1.0

10. CONCLUSION

We may conclude the paper by pointing out that, this CASE tool will play an important role to the software developers and learners to use and understand the utility of the CASE tool in today's complex software projects. Also, as we mentioned earlier, interested educational institutions and organizations may contact the author for the CASE tool for their usage.

11. FUTURE WORK

SRS BUILDER 1.0 CASE tool has been tested and validated properly. In future, we propose to enhance the capabilities of the present version by appending the functionalities to design the various UML diagrams.

12. REFERENCES

[1]. Ilvari, J. (1996). '*Why are CASE Tools Not Used?*". Communications of the ACM, 39:94-103.

[2]. Jarzabek, S. and Huang, R. (1998). '*The case for User-Centered CASE tools*", Communications of the ACM, 41(8): 93-99.

[3]. Kemerer, C. F. (1992). "*How the Learning Curve Affects CASE Tool Adoption*". IEEE Software, 9, 23-28.

[4]. Mall, Rajib. "*Fundamentals of Software Engineering*", Second Edition, PHI.

[5]. Mandal, Ardhendu. "*BRIDGE: A Model for Modern Software Development Process to Cater the Present Software Crisis*", Proceeding, PP: 494-500, IEEE International Advance Computing Conference (IACC 2009), Patiala, India, 6–7March 2009, ISBN-978-981-08-2465-5

[6]. Roger S. Pressman, "*Software Engineering: A Practitioner's Approach*", Mc-Grawhil, Sixth Edition, 2005.

 ISSN 0973-7529 ISBN 978-81-904526-9-4

2009 IEEE International Advance Computing Conference (IACC 2009)
Patiala, India, 6–7 March 2009

BRIDGE: A Model for Modern Software Development Process to Cater the Present Software Crisis

Ardhendu Mandal
Lecturer, Department of Computer Science and Application, University of North Bengal
Raja Rammohanpur, PO-NBU, Dist-Darjeeling, West Bengal, Pin-734013, India.
am.csa.nbu@gmail.com

Abstract-*As hardware components are becoming cheaper and powerful day by day, the expected services from modern software are increasing like any thing. Developing such software has become extremely challenging. Not only the complexity, but also the developing of such software within the time constraints and budget has become the real challenge. Quality concern and maintainability are added flavour to the challenge. On stream, the requirements of the clients are changing so frequently that it has become extremely tough to manage these changes. More often, the clients are unhappy with the end product. Large, complex software projects are notoriously late to market, often exhibit quality problems, and don't always deliver on promised functionality. None of the existing models are helpful to cater the modern software crisis. Hence, a better modern software development process model to handle with the present software crisis is badly needed. This paper suggests a new software development process model, BRIDGE, to tackle present software crisis.*

I. Introduction

Now a day, computers running with special purpose application software are being used as an extensive aid to solve complex problems almost each and every place starting from gaming to engineering, industries applications, scientific research and different allied fields. These special purpose softwares are some times unique and distributed in nature with higher degree of complexity. Developing such complex software is not so easy because of the different constraints. Our existing software models do not provide adequate flexibility to be applied for such large and complex projects. So we must have a better software development process model that will help to overcome these challenges.

II. Usage of Different Process Models: A Survey Report

The result of the survey carried out by Dr. Jon Holt [3], related to current practice in software engineering reveals the percentage of usage of different types of software development lifecycle models (SDLC) in practice. The result is shown below in *Figure 1*. Although different organizations do use different lifecycle models, but from the above data it is clear that *a large part of industries (22%) do not use any lifecycle model at all!* The BIG question here is why these organizations do not follow any life cycle model?

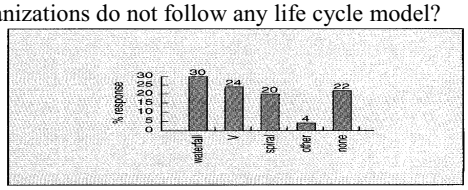


Figure 1: Use of different SDLC models in Practice [3]

The probable answer is, either no lifecycle model is suitable for their projects or they don't find it useful. In either case, it means the existing models lacking suitability. Hence, we need to improve the suitability of these models so that it can be used in practice.

III. Characteristics of good Software Development Process Model

Any software development process model should have the following characteristics [2] for quality software development:

i. *The project goal reflection* i.e. the process model must reflect the project development goals.

ii. *Predictability i.e.* it must be able to forecast the out put of the project following the model prior to project completion.

iii. *Support testability and maintainability* i.e. the process model must focus on reducing the cost, effort of testing and maintenance.

iv. *Support change* i.e. the process model must handle the necessary changes.

v. *Early Defect Removal,* because the delay in error detection increases the costs to correct them.

vi. *Process improvement and feedback* i.e. each project done using the existing process model must feed information back to facilitate further process improvement.

vii. *Quantitative progress measurements* i.e. the process model at any point must give a quantitative measurement of the progress attained.

viii. *Support of process tailoring* in special situations at necessity.

IV. Nature of Modern Software Projects

The earlier software projects were of limited scope with relatively less complexity and smaller size. In contrast, the modern software has *wider scope, higher degree of complexity and larger size with better quality, portability* and *scalability* requirements. Some times, the modern software has *to work with some existing legacy system*. Developing such system are more challenging because of the *inter-operatability* and *dependency* factors. The modern real-time systems have lots of *critical issues* such as *time and space complexity* requires to be addressed. Tremendous hardware development rate has brought us towards the system-on-chip (SOC) era. In such systems, the software has to work in coordination with the particular hardware. Developing such systems are more critical because of the hardware *constraints*. As result of advancement in network technology, more often systems are becoming *web based* and *distributed* in nature. In conclusion, the modern softwares are different in various respects from the earlier softwares.

V. Modern Software Crisis:

Software Crisis may be loosely defined as the *problems associated with the software development process*. Among a lot, a few critical software crisis with modern software development are listed below [5,8]:

i. Larger *size.*
ii. Increasing *complexity.*
iii. Higher development *cost*.
iv. The *delivery* challenges i.e. *late* system delivery.
v. The *trust* challenge. How much can we trust on system operations?
vi. *Incorrectness*: Not satisfying the client needs exactly.
vii. Poor *quality.*
viii. Poor *productivity.*
ix. The *heterogeneity* Challenges i.e. inter-system coordination problem.
x. Demand of *reusability.*
xi. *Modularity*.
xii. *Maintainability*.
xiii. *Integration* problem.
xiv. *Scalability*.
xv. *Portability*.
xvi. *Change* Management.
xvii. *Risks* associated with software development.

VI. Trends in Modern Software Development

Recently, lots of new approaches are being used at practice to overcome the modern software crisis. Some recent trends in modern software developments are listed below:

i. Component based software development.
ii. Software reuse
iii. Aspect oriented software development.
iv. Service oriented software development.
v. Multi-Tired Software Design.
vi. Object Oriented Software Development.
vii. Standards practices.
viii. Use of CASE tools.

VII. Reasons for failure of Traditional SDLC Models: The Shortcomings

After analysing the existing SDLC models, the shortcoming of these models may be broadly summarised as follows:

i. *Non-Involvement of the client* over the entire project development.
ii. *Lack of better understanding* of the system requirements.
iii. *Lack of communications* among the team members.
iv. *Lack of project management* controls over the entire development period.
v. *Overlooking verification* activity
vi. *Insufficient documentations*.
vii. *Lack of configuration management*.
viii. *Non importance to component based* software *development and*
ix. *Poor support of component reusability*.

Directly or indirectly, the above reasons are the real causes of the various software crises. I have tried to address these causes of software crisis in my proposed model discussed shortly.

VIII. Need of Modified Process Model

Although, tailored traditional software development process models are being used since a long time, but these are not good enough at practice. Hence, we are in search of a new software development process model that will adopt and encourage these modern practices. In the forth-coming section a rather novel software development process model-BRIDGE, is proposed and discussed that attempts to encourage the modern software development trends. As well said by David Norton, research director at Gartner "I do not feel waterfall development was bad. It's given us a lot of software over the last 30 years, but I think its time is up"[1].

IX. BRIDGE: The Proposed Model for Modern Software Development Process

After analysing the importance of all the recent software development trends, at attempt is taken to develop a rather new and novel software development process model that adopts the modern software development trends and practices. The so named BRIDGE model is the result of such an attempt, which is elaborated over the following sections. The schematic diagram of the BRIDGE model is given in *Figure 2*.

A. BRIDGE Process Model Description:

Unlike the other process models, the BRIDGE model consists of several phases with distinguished objectives that are discussed in the following section briefly:

i. Phase1: Requirement Analysis, Verification and Specification

The *objective* of this phase is to *identify the exact requirements* from the client using different techniques and to specify them in a document for future use after verification. During *requirement gathering,* the analyst extracts the system requirements from the client. In practice, it is really a tough job for the analyst to extract the requirements from the client, as the clients are unable to identify and express the exact requirements prior experiencing the system practically. The gathered requirements required to be analyzed for removing the redundancy, incompleteness, inconsistencies, anomalies etc. This phase is often called the *requirement analysis phase.* Finally, the verified requirements are to be *specified* in a document called *Software Requirements Specification (SRS)* and stored for future use. This phase is often called *requirement specification* phase. This SRS document may serve as the agreement document between the client and the company and becomes the baseline for proceeding to the next phase.

ii. Phase2: Feasibility Analysis, Risk Analysis, Verification and Specification

The *objectiv*e of this phase is to *analyze the suitability* of the project in respect to different project attributes to check the different suitability aspects among the alternatives. After carrying out the analysis, the optimal solution is selected. At this stage the project cost estimation has to carry out. The different feasibility i.e. *economic feasibility*, *technical feasibility*, *operational feasibility* has to carry out to manage the different system constraints. Some times, the result of the different feasibility analysis may contradict. In such cases, necessary changes, modification and/or negotiation may have to do in the project upon consulting the client if the project is not cancelled. Finally, after verification the result of the feasibility analysis has to be specified in a document called *feasibility report and to be kept for future reference.* Beside feasibility analysis, at this phase the different project risks have *to be identified, analyzed* and specified in the *risk specification document.*

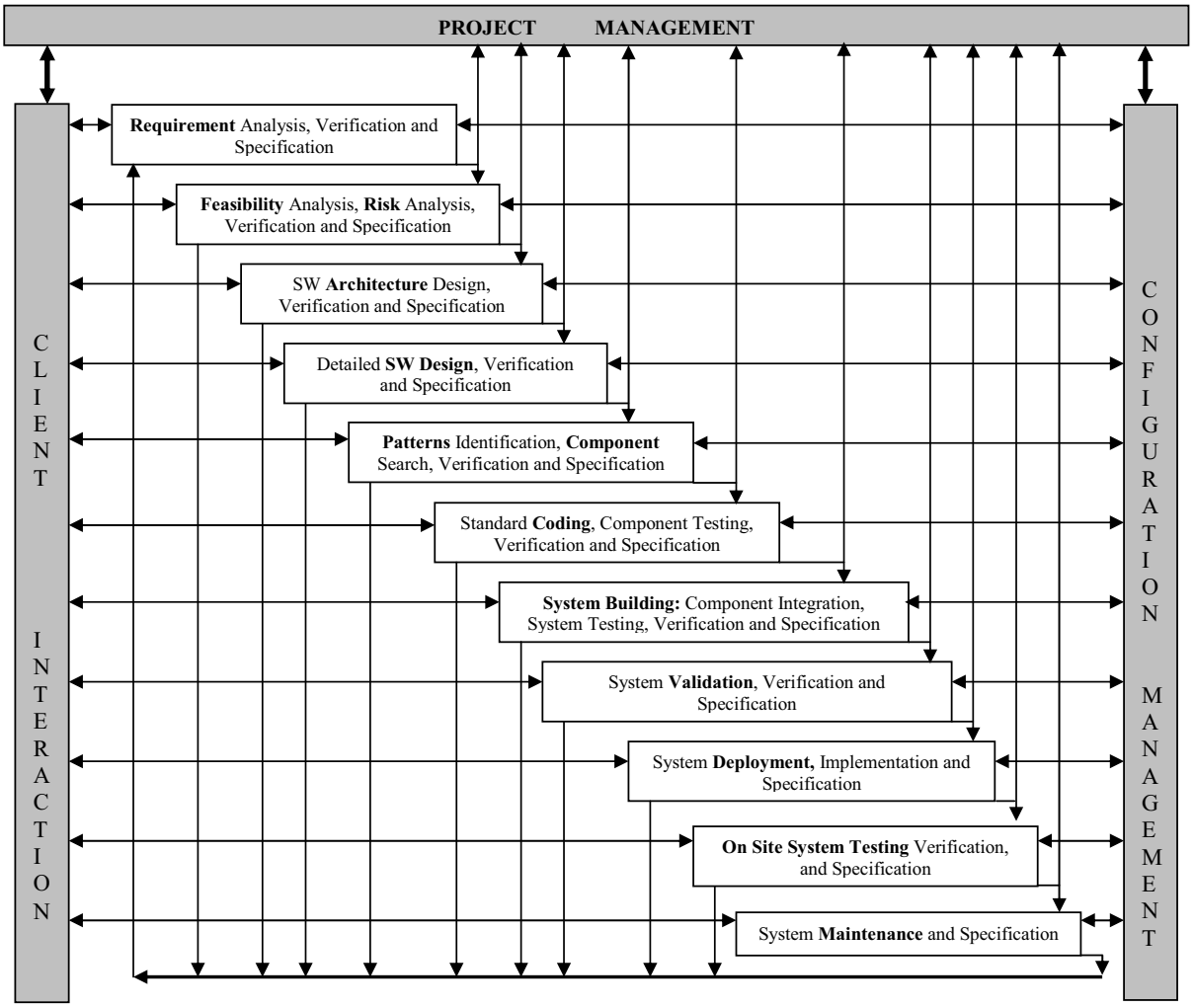


Figure 2: BRIDGE Software Development Process Model

iii. Phase 3: Software Architecture Design, Verification and Specification

Once the project is confirmed, we must design the software architecture. Software architecture design is a *high-level design activity* and relatively a recent trend in industries after understanding its importance. We may consider software architecture as *abstract design* of the complete system. The *objective* of software architecture design is to *identify the sub-systems, building blocks or the components* of the system along with their *communication interfaces* expressing their *external behavior* to improve the project understandability and to communicate with the different stakeholders. The architecture design should reflect the functional requirements specified in the SRS document. Once the software architecture is designed, the architecture design must be *verified* to check whether it conforms the system requirements correctly or not. The verified software architecture design is specified in a *software architecture design document (SADD)*. It must be clear that implementational issues are not considered while designing the software architecture.

iv. Phase 4: Detailed Software Design, Verification and Specification

In this phase, the *detailed design* of the system has to be prepared conforming the software architecture designed during the last phase. Software design is basically a *low-level design* activity *keeping the implementational issues in mind*. The *objective* of this phase is to prepare the *modular design* of the system that can be *directly implemented using some programming language*. The *data structure and algorithms* are also to be developed in this phase. The verified software design specified in a document named as *software design document (SDD)* that will be used in the other development activities later.

v. Phase 5: Patterns Identification, Component Search, Verification and Specification

In general, a system consists of a set of sub-systems, so called *components*. If we analyze any problem, we may find some components common in different projects *representing some general structures of a system.* These common components are sometimes called *patterns.* The *objective* of this phase is to *identify these patterns*. But, to use these pre-developed components efficiently in our system, the *system must be designed keeping this objective* in mind and the designer should be well aware of the available components in the component library. From the architecture design, we must

be able *to identify the components* and then it must be *searched* in the component library to find a suitable *component match*. Before moving to the next phase, we must verify the current phase properly and specifying in a document called *component specification document (CSD*) for future use.

vi. Phase 6: Standard Coding, Unit Testing, Verification and Specification

All the components identified during the last phase may not be available in the component library. The *objective* of this phase is *to write program code for the unmatched components*. Often, a few unmatched components may work as desired just with a suitable added interface. In those cases, the benefit analysis must be done to take the decision whether to develop the interface only or the unmatched components from the scratch. *The unmatched modules must be coded properly following the standard coding guidelines and practices* laid down by the organization itself or the available standard conventions as per the organization interest. These newly developed components must be tested thoroughly since these components are going to be used in several systems at different times. Such testing is called *unit or component testing*. The components taken from the component library together with the newly developed components should be sufficient enough to build the whole system. The newly developed components may be added in the component library for future use if it looks justifiable. After verifying and specifying the phase properly, next phase can be started.

vii. Phase 7: System Building: Component Integration, System Testing, Verification and Specification

Once all the individual *components* are gathered, it's the time to *integrate* these to build the whole system preferably following the bottom up approach. Hence, the *objective* of this phase is *to build the whole system by integrating all the components*. However, it is not necessary that, after integrating the pre-tested components successfully, the integrated system will work correctly. Various types of problems such as type mismatch, number of parameter mismatch, return type mismatch etc. may arise. Hence, there is a need to test the integrated system at different level of integration. This is called *integration testing*. Now, the complete build-up system has to be tested thoroughly using the different testing techniques to check the correctness of system functionality. The testing at this topmost level is termed as *system testing.* After performing the different testing, the corresponding *test report* has to be prepared for use during system validation and maintenance activities. Finally, the phase verification is to be carried out prior moving to the next phase.

viii. Phase 8: System Validation, Verification and Specification

Merry successful verification of the system doesn't ensure the fulfilling of all client requirements! By successful verification of the system, we can only ensure that whatever the functions are implemented in the developed system do work correctly, but does it mean that, all the function required by client are implemented in the system? **No**. The ***objective*** of this phase is *to check* whether *all the functional requirements* as specified in the SRS document specified by the client *are exactly included the system or not.* There must be one to one correspondence between the functions in the SRS document and function supported by the system. Performing this activity is called ***system validation***. Not only the system functionality but also the quality of the system has to be validated. Unlike the other phases, at the end, this phase to be verified and the out come of the system validation activity are to be specified in a document called ***validation report*** and stored for further use.

ix. Phase 9: System Deployment, Implementation and Specification

Once the system is validated, now it's the time to deliver the system to the client and implement the system at client site. Again, some more changes may be required to accommodate and adjust for proper functioning of the system. *Delivering the system to client should not be taken as a formality! Ultimately you- the developers are not going to use the system, but the users definitely.* Until the users are not able to use the system effectively and efficiently, developing the system remains purposeless. We must facilitate the user to understand and feel comfortable with the system at use. There are basically two tools for this purpose. First, the *documentations* and second, *training*. Necessary training has to be provided to all different categories of users within their operational scope. The user refers to the documents to solve problems at any point of time during the system use. The ***objective*** of this phase is *to deliver, implement the system at client's work-site and train the users, if necessary.* After verification of the phase necessary documents are to be prepared and retain.

x. Phase 10: On Site System Testing Verification, and Specification

Although, system testing is completed prior to system implementation, but due to different environmental changes and other reasons, the system may not function correctly at the work-site. Hence, after implementation, the system needs to be tested at work site too. This testing is called ***on-site system testing***. The ***objective*** of this phase is to *check the system performance at work-site*. Finally, the *on-site system testing report* has to be prepared and to be retain after the phase verification. At this point, the current system is at work.

xi. Phase 11: System Maintenance, Verification and Specification

Merry successful system implementation and functioning is not the end job. There is a well saying that *no software is correct at all.* Moreover, Lehman's first law related to software says, *"Software product must change continually or become progressively less useful"* [5]. Software Maintenance denotes any changes made to a software product after it has been delivered to the client. Maintenance is a continuous process over the software life cycle. The ***objective*** of this phase is *to provide the post delivery services to the system for its desirable functioning*. Maintenance support is to be provided to retain and improve the system quality over its lifetime. The maintenance may be of different types i.e. *corrective maintenance, adaptive maintenance, perfective maintenance and preventive maintenance* [5,6]. Finally the *maintenance report* is to be made periodically and kept for future reference.

It should be clear that deliverables from any phase might be given as input to the other phases if needed.

xii. Phase 12: Configuration Management

As we have seen, *system requirements always change during system development and use*. Accordingly, these changes have to be made in associated documents and dependable. Finally, these changes are *to be incorporated into new version of the system.* Hence, it is clear that, the deliverables from different phases are to be maintained for future use. As we have seen over the past discussion that, *from each phase different documents are produced and need to be kept properly for future use.* The patterns are even to be kept in such a way that, on demand we must be able to identify, search, and locate all these components for adaptation. Hence, we need to have an efficient *document and component keeping system*. If there is no proper management and control over these changeable particulars, then it is very tough to incorporate these necessary changes in the following version of the system. *The means by which the process of software development and maintenance is controlled is called configuration management.* The ***objective*** of the configuration management is the development of procedures and standards for cost effective managing and controlling the changes in the evolving software system aiming *to keep track of all the important* deliverables obtained from different phases.

xiii. Phase 13: Project Management

Unlike the configuration management activity, the project management activity has to be carried out in parallel with all the other software development phases. While developing software, we need to carryout some management activities that are part of project management. The ***objective*** of this phase is to *perform the project management activities including project planning, monitoring, controlling, directing, motivating and coordinating.*

X. Analysis of the BRIDGE Model

The in-depth study of the BRIDGE model discloses a lot of information that may be used to analyze the model. These are briefly discussed below:

A. Findings from the Study of BRIDGE Model:

The findings from the BRIDGE model are listed below:

ix. It *involves the client* over the entire development life cycle activities.
x. It keeps continuous *communication with the project management* team.
xi. It explicit *verification of individual* phases.
xii. Separate software *architecture design* phase.
xiii. Separate *system deployment* phase.
xiv. Separate *on-site system testing phase.*
xv. Supports *components based* software development.
xvi. It emphasizes on *standard coding*.
xvii. It considers *configuration management* as a separate activity.
xviii. It forces to *specify* all the phase deliverables.
xix. It explicitly instructs to *validate the system.*

B. Impact analysis of findings from BRIDGE Model Study

In this section, impacts of the findings from BRIDGE model studies on the project goal are analysed distinctly.

i. Impact of continuous client involvement:

It is experienced that, as the system is more studied and analyzed over the time, the client specifies more new requirements. Satisfying these requirements, *client satisfaction* and *software quality* are improved with great impact on both *project* and *organizational goal*. Moreover, involving the client over the entire SDLC *project risks can be alleviate* up to a significant extent. By means of continuous client involvement, this model can embed the *prototyping paradigm of software development.*

ii. Impact of continuous project management team involvement:

The impact of involving the project management team over the SDLC model may facilitate *effective project management activities* such as *project planning, progress monitoring, project controlling*, r*isk management, Motivation* and individual *performance analysis* used for organizational and personal appraisal.

iii. Impact of explicit verification activity:

By verifying the individual phases indirectly the *phase entry and exit criterion* may be satisfied which reduces the error occurrence rate in the later phases. This may even overcome the well-known *99% complete syndrome problem*. Verification helps in *early error detection and correction reducing total development cost having direct impact on software testing, quality control and timely product delivery.*

iv. Impact of software architecture design:

Software architecture is the key framework better project understanding and communication with the various stakeholders. Software architecture has a profound influence on organization functioning and structure [4]. Designing the software architecture has the direct impact on the software quality attributes such as *performance, security, safety, availability, maintainability, scalability, productivity, cost, effort and timely product delivery.*

v. Impact of separate system deployment phase:

It directly maps the *environmental view supported in UML*. There is a very poor practice of considering system delivery as just a formality. Proper training must be given to the users for efficient and effective system use. More over it helps to handle all software crisis related to product deployment improving the software quality.

vi. Impact of separate on-site system testing phase:

The on-site testing helps to improve system quality and client satisfaction reflecting the long-term goal of the Organization.

vii. Impact of component based software design:

The component based software design helps in achieving better *software maintainability, reusability, productivity and quality reducing total development cost and effort.*

viii. Impact of following standard coding:

Following standard coding practices and conventions have remarkable impact on *better understanding* of the code written by others *reducing efforts in error isolation and system testing improving the maintainability, quality of the software. It does encourage good programming practices.*

ix. Impact of configuration management activity:

Configuration management activities improve different documents and components management. It does facilitate component repository and reusability reducing total development cost and efforts improving the software quality and increasing organizational assets simultaneously.

x. Impact of document specification:

The different specified documents facilities *better system understanding* leading to *ease error handling*. These are the

means of communication among teammates and stakeholders. It helps in *reduction of testing and maintenance efforts*.

xi. Impact of system validation:

System validation ensures correct system functionality by error detection achieving the goal of better quality software development. Finally, it increases degree of client satisfaction attaining long-term project and organizational goal.

XI. Validating the BRIDGE Model in Support of Goodness Criterion

The proposed BRIDGE model does satisfy almost all the goodness criterion [2] of a good software development process. In this section, I discuss the supporting issues for validating this model against the individual goodness criteria.

i. Support towards project goal reflection

As per the definition of software engineering given by Stephen Schach [7], the goals of software project are:

a. Developing ***quality*** software.
b. Developing the software ***within budget***.
c. Delivery of the software ***within time***.
d. ***Satisfying customer*** requirements.

By focusing on the phase verification and validation activities, and recommending software testing at different levels, this model reflects the goal of developing *quality software*. Again, specially performing economic feasibility analysis and involving the management over the process, the model reflects the goal of developing the software *within budget*. On stream, by involving the project management team over the entire process development model, this model puts focus on proper management control to follow the time constraints on the project development. Finally, by means of client involvement over the complete software development process, the BRIDGE model achieves the goal of *customer satisfaction*.

ii. Support of Predictability

The software architecture is the best document to predict the different project parameters. Having a separate software architecture phase and risk analysis, this model achieves the predictability criteria.

iii. Support of testability and maintainability

Emphasizing on component based software development and component reusability concept, this model highlights the testability criteria. In addition, designing the software architecture gives the foundation for meeting maintainability criteria with a separate phase related to software maintenance.

iv. Support towards change

Designing software architecture and by supporting maintainability, this model achieves the change management criteria directly with consistent support from configuration management.

v. Support of early defect removal

By involving the customer over the entire development process, it is possible to detect errors at earliest and performing verification activity following each phase ensures early defect detection and removal.

vi. Support of process improvement and feedback

During the configuration management activities, all the prepared documents and reports are stored. Project completion analysis report with the available documents and the reports from configuration database, can be used to judge and identify the activities needing process improvement and applying the same in the next project. Customer comments and recommendations can be used as the feedback for further process improvements.

vii. Support of Quantitative Progress Measurement

Directly, each phase indicates a milestone towards the project completion. All the deliverables from various phases of this process model can be used to measure the progress of the work completed.

viii. Support of Process Tailoring

Since the process activities are decomposed in several phases, at necessity, more than one phase can be combined and any phase can be further decomposed into sub phases or even might be dropped depending on the project characteristics. Hence, it may be concluded that, the BRIDGE model satisfies all the desired characteristics of a good software process model.

XII. Suitability of the BRIDGE Model

This model can be used to both simple systems as well as complex systems. It supports the object oriented, component based software development paradigm. By process tailoring, this model also can be applied to develop any software projects that are directly unfit to the actual model. Hence, the suitability of the BRIDGE model for any modern software development is justified and may be recommended for any kind of software project development.

XIII. Limitations of the BRIDGE Model

Along with the strong suitability, this model has some limitations as pointed down below:

a. Non-considering the implementational issues.
b. Abstracts the different techniques to be used in different phases.
c. Required to be validated by industrial practice.
d. It doesn't consider professionals skill level.
e. The BRIDGE model seems to be complex.

XIV. Naming Significance: BRIDGE

The schematic diagram of the proposed model looks like a bridge. In a bridge, the entire load is on the bridge floor, but this load has distributed over all the pillars for its survivals. Directly the project pressure is on the "Project Management" and this pressure has to be distributed over "Client Interaction", "Configuration management " and other the phases indirectly- the pillars of the model. Keeping this point of view the name, BRIDGE, is given and justified.

XV. Conclusion

After the complete analysis, it can be conclude that if the BRIDGE model is followed to any software project development, most of the software crisis may be overcome up to great extent delivering the fully functional system with better quality within time and budget achieving the true goal of any software project development.

References

[1] Gerg Goth, "*Software-as-a-service: The Spark That Will Change Software Engineering?*" IEEE distributed systems, July 2008.

[2] Jalota P., *Integrated Approach to Software Engineering*, Narosa, Third Edition. 2006.

[3] Jon Holt, *Current practice in software engineering: a survey*, Computing and Control Engineering Journal, 1997.

[4] Joseph F. Maranzano, Sandra A. Rozsypal and Gus H. Zimmerman, Guy W. Warnken and Patricia E. Wirth, *Architecture Reviews: Practice and Experience*, IEEE Software, 2005.

[5] Mall R., *Fundamentals of Software Engineering*, PHI, Second Edition, 2008.

[6] Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, Mc-Grawhil, Sixth Edition, 2005.

[7] Stephen Schach, *Software Engineering*, Vanderbilt University, Aksen Association, 1990.

[8] Pfleeger S. L., *Software Engineering: Theory and Practices*, Pearson Education, Second Edition, 2007.