Asian Journal of Information Technology 7 (12): 510-515, 2008
ISSN: 1682-3915
© Medwell Journals, 2008

Process Model for Building Quality Software on Internet Time

Godspower O. Ekuobase, Francesca A. Egbokhare and Veronica V.N. Akwukwuma
Department of Computer Science, University of Benin, Benin, Nigeria

Abstract: The competitive nature of the software construction market and the inherently exhilarating nature of software hinged the success of any software development project on 4 major pillars: time to market, product quality, innovation and documentation. Unfortunately, however, existing software development models are either a bunch of ponderous bureaucratic serial processes or unstructured set of agile processes. While the former ensures appropriate documentation and quality of software product, it undermines the criticality of innovation and time to market, the latter holds time and innovation in high premium but treats the issue of quality and documentation as secondary. We therefore, lack but need a software process model that holds these 4 success factors in high esteem. This study proposes, one such model-the MULTIPARL model. The MULTIPARL model is a structured time driven model that ensures a software product is constructed and delivered as increments of partial functionality. In particular, the MULTIPARL model stood out as the most appropriate process model for web-based application development when it was subjected to the Onibere's process selection criteria alongside some active software process models.

Key words: Software, software development, software process, MULTIPARL model, development project

INTRODUCTION

Software engineering, a child of crisis, is still in crisis. The continuous influx of radically distinct development paradigms and process models into the software engineering domain is a pointer to the failures or frustrations associated with existing ones. These failures and frustrations have taken us back to where we started unstructured set of development activities, which was called cut and nail or code and fix and is now called agile method. Whether software development is open or closed; our (users and developers) expectations are yet to be fully meant (Paulson *et al*., 2004). The expectations of software engineers and users have always been early delivery of easy to use dependable software that is economical.

A formal development structure is imperative to meet these expectations (Onibere and Ekuobase, 2006) and now that several such structures exist, why the embarrassing result on software products as unveiled by Paulson *et al.* (2004). Ekuobase (2006) and Onibere and Ekuobase (2006), reported increasing complexity in the engineering of software products as well as the radically distinct nature of software from other engineering products as a notable source of the failures and frustrations associated with the engineering of software products. An appreciation of these facts account for the birth of the agile approach to software development (Alliance, 2001; Abrahamsson *et al*., 2003; Aoyama, 1998; Cockburn, 2002) but the pessimism about agile methods (McBreen, 2003) is critical. Miller (2001) and Ekuobase (2006), say the agile processes will not do better than its predecessors. A little wonder why Boehm (2002) and Boehm and Turner (2005) demands for a mix of the rigid and agile software development processes in the construction of software. The rigid or traditional software processes (Sommerville, 1996; Scacchi, 2001; Ekuobase, 2004) is a bunch of ponderous bureaucratic serial processes that can only be suitable for the design, development, deployment and maintenance of big and slowly adapting systems (Cusumano and Yoffie, 1999; Griss and Pour, 2001). Worst still, the present day software market which is driven by innovation and time-to-market cannot tolerate these slow and rigid traditional processes. The agile processes on the other hand are characterized by modularity, iteration, parsimony, time-bound, adaptation, incremental delivery, convergence, people-centric and collaboration (Miller, 2001) and can best be described by the agile manifesto, which was signed in February 2001:

> we are uncovering better ways of developing software by doing it and helping others to do it.

Through this research, we have come to value

- Individuals and interactions over processes and tools

Corresponding Author: Godspower O. Ekuobase, Department of Computer Science, University of Benin, Benin, Nigeria

- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following plan

That is while there is value in the items on the right, we value the items on the left more (Alliance, 2001).

Items on the right define the traditional processes, while items on the left define pure agility. In practice, items on the right are difficult to adequately consider, if we actually want to go agile. As long as agile methods treat formalized processes, planning and documentation as secondary issues as evident in the Agile Manifesto, our tomorrow will experience worst crisis in the domain of software development. The proponents of the agile methods are only having transient successes today because they were trained in the traditional way and still have the traditional spirit in them, which their so called better ways only help to modify but by the time we have a new generation of software developers trained and nurtured the agile way and unawares of the traditional approaches we will experience a worst crisis in software development than ever experienced; for agile methods can easily degenerate into an unstructured informal process as the code-and-fix (Sommerville, 1996; McBreen, 2003), with all its inherent problems.

Consequently, agile methods will need a formal development process or structure that can enforce planning and comprehensive documentation, which are critical to the dependability of a software product. This is the essence of this study to propose one such Agile software process model that enforces planning and comprehensive documentation.

SOFTWARE CONSTRUCTION MARKET

First-to-market, innovation, partial functionality and incremental delivery, usability, product quality and documentation.

First-to-market: The nature of the software market is such that for any product to be useful, competitive or more interesting, it must get to the market first (Cusumano and Selby, 1997; Cusumano and Yoffie, 1999; Baskerville *et al.*, 2001). The software product that reaches the market first has the greatest promise for capturing a wider explosion of interest. This applies not only to new systems but also to new product features or concepts in existing systems.

Innovation: This ensures continuous and competitive relevance of a product in the market. It is not over when a product is the first in the market, advanced features must be continually developed and added to remain competitive. Most of the notable successes in the software domain were products of innovation (Ekuobase, 2006).

Partial functionality and incremental delivery: Those that waited to meet all the anticipated requirements of a software product before shipment are today frustrated in the software construction market (Cusumano and Selby, 1997; Baskerville *et al.*, 2001). The market is such that you get to the market as quickly as possible with what is available, interesting and useful and bring in innovations, bug fixes, performance and reliability improvements with time though quickly too; before these lapses are exposed by rival software products or disaffection is created among the users of the product. With this, more money is made, while one maintains a competitive market position.

Usability: We like it or not, the user is the lord of the software construction market (Ekuobase, 2006) and decides, which product remain relevant or competitive in the market. Unfortunately, however, their judgment is based more on ease of use and the visible features of the software product than on inert software qualities. The reality is that a less powerful software product with excellent usability attribute can dominate the most powerful software product with poor usability features in the market.

Product quality: The quality of a software product remains the most dominant capability to remain relevant in the software market. Over the years, however, software quality could not assume its priority position due to the dynamics of the market. Successful developers under played software quality by delivering products to the market as releases; non-functional features as scalability, reliability and security were gradually introduced or improved with subsequent releases (Baskerville *et al.*, 2001). These developers understood some important facts that the test of quality is a function of use time and only noticeable by a minority for only a few instances within a short period of use. They also understood that users feedbacks are critical to the overall product enhancement.

Documentation: Documentation does not only enhance conceptualization of product and its feasibility but also allows for proper coordination, continuity and use of staff, processes, tools and software product. Actually, most aspects of documentation from requirement specification to design and to other coding patterns as indentation and comments are purely technical and hidden. The impact of documentation though critical is not felt in the market. Even the user manual is becoming less useful as a result of the similarity and simplicity of interfaces. Unknown to a few, documentation aids software innovation, quality and the time-to-market in the form of reuse, software components and patterns.

THE NATURE OF SOFTWARE

Software is not like other engineering products as bridge, car or computer hardware (Sommerville, 1997; Snowdon, 2003). Though, we knew early that software can not be completely specified or built (Basili and Turner, 1975), we are yet to appreciate that software is one product with fluid nature. The rate of change is a function of need, domain, or drive. Today's society is driven by speed and comfort and software in response to societal drive is today driven by ease of use and time-to-market. Yesterday, it was driven by need and quality (Ekuobase, 2006) and tomorrow, it may be something else. What is clear already is that its behaviour is always consistent with that of the society. Also worthy of note is the expected lifespan of software, it is supposed to out live several generations; software though fluid has no tear and no wear. This singular nature of software makes proper documentation indispensable; at least for continuity in maintenance.

Considering the nature of software and the dynamics of its market, we see that agile software development process is not a replacement of the traditional processes but has excellent features that can be used to upgrade the traditional processes to meet modern challenges. These features are basically speed and innovation (Highsmith and Cockburn, 2001). A software process model will however, not accommodate quick and regular innovations without it being flexible. In the following study, we present a software model that incorporates the features of the old and the new based on the nature of software and the dynamics of the software construction market.

THE PROPOSED SOFTWARE PROCESS MODEL

The proposed software process model, the MULTIPARL (MULTIple-PARalleL) model (Fig. 1) is a disciplined time driven model that enforces incremental delivery in the form of multiple releases, feature prioritization and slip, fluid specification and design, user participation, process adjustment and technology insertion that are essential for rapid development and constant maintenance or enhancement of software systems. The model is an infinite process and consists of independent releases running in parallel with each successor release accommodating feedbacks from predecessor releases at convenient phases of its lifecycle. The disciplined approach of this model is inherent in the waterfall nature of each release, while flexibility (agility) is enforced by feedbacks across releases, its fluid specification/design and slip of features working against time to appropriate phases of successor releases. Feedbacks within releases allow for change. The different releases of this model should be handled by different development teams, which may not necessarily start at the same time. Two to three teams may be ideal; if any team completes a cycle, they pick up a fresh release.

As shown in Fig. 1, each release consists basically of four phases: preparation, build, Quality Control (QC) and use phases. The preparation phase in turn consists of three sub-phases or activities: requirement definition, conceptualization and prioritization and fluid specification and design activities.

Preparation phase: This phase enforces formalized planning and (external) documentation constrained by deadline. Hence, the activities of feature prioritization, fluid specification and design that allow for feature slip or carryover to successor releases. The requirement definition sub-phase of the first release however, becomes requirement review in subsequent releases. This phase accepts feedbacks from any phase of predecessor releases or the immediate successor phase (build phase) within release and sees if they can be incorporated or slipped to successor releases. The sub-phases of this phase can all be simultaneously or repeatedly implemented but the outputs of this phase are comprehensive requirement and design documents. Feedbacks can leave this phase to other preparation phases of successor release.

The requirement definition activities determine, analyze (or review) and define the requirements of individual systems release. The conceptualization activities can be conceptual or concrete (prototyping, say). In any case, it is aimed at determining/demonstrating the criticality and feasibility of requirement features. This set the stage for feature prioritization, specification and design. Feature prioritization brings preference into the requirements based on the dynamics of the market and the time at hand such that requirements with low preference may slip to successor releases. The final set of activities

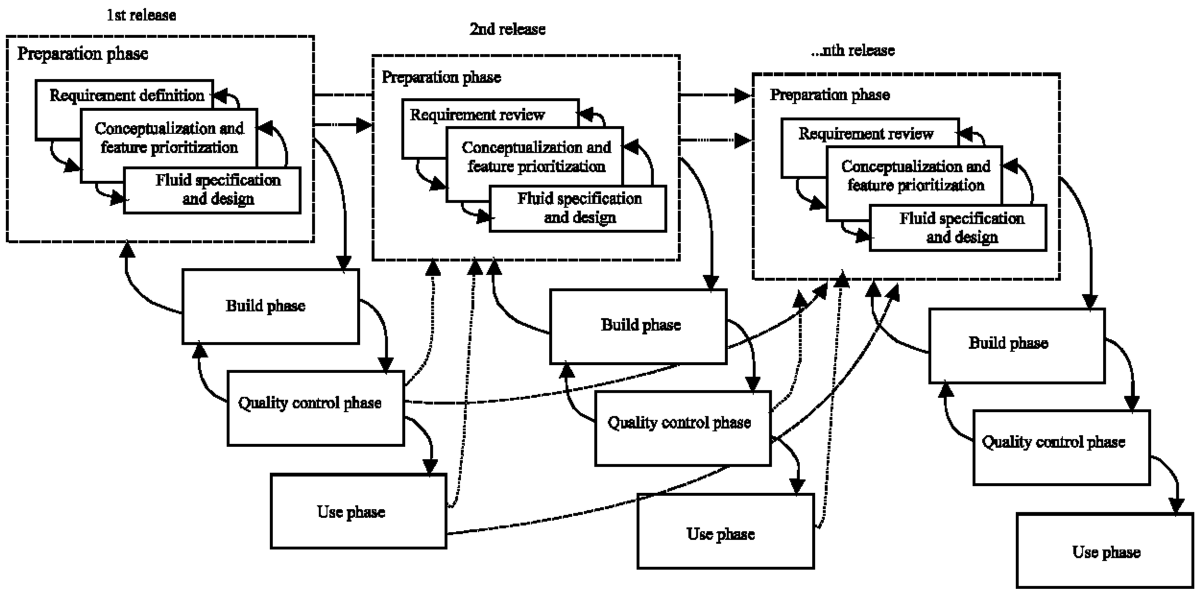

Fig. 1: MULTIPARL model

for this phase is fluid specification and design. By fluid, we mean design in anticipation of change. The model insists on comprehensive design document for each release. This is possible particularly with the use and continuous development of software patterns (Fowler, 1997, 1999, 2002; Alur *et al.*, 2003).

Build phase: This phase is concerned with the implementation of the design document for each release. It includes the activities of coding, component selection and integration, sub system integration and testing. This phase does not accept feedbacks from predecessor releases except from the quality control phase within release and is not allowed to communicate with external releases. This constraints help check the excesses of flexibility in agility. The output of the build phase is working software.

Quality control phase: This phase ensures software is built to specifications and approved standards. Its activities also include quality assurance and acceptance tests. It is responsible for passing the software for market. It does not accept feedback from any phase or release but can send feedback up within release and to successor releases. The end product of this phase is certified working software.

Use phase: This phase involves the actual use of the certified and released software product. Here, feedbacks (in form of bugs or user complaints/preference) are collected and used to enhance/modify future releases. It cannot accept feedbacks and does not send feedback up within release. This phase ensures full user participation in software development particularly after the initial software release.

THE MULTIPARL PROCESS MODEL IN PERSPECTIVE

The proposed MULTIPARL model though yet to be put to real life use did not fall short of expectations in an experiment with some selected active software process models using the Onibere's software process selection criteria for Web based application development (Ekuobase, 2004; Onibere and Ekuobase, 2006; Ekuobase and Onibere, 2007). Table 1 is a recap of the result of the experiment to establish the supremacy or otherwise of the MULTIPARL model over currently active software process models with details in Ekuobase (2004). From Table 1, we see that this model is particularly strong for medium to large scale software development projects.

Technically, one may not easily distinguish this model from the evolutionary model but note that the evolutionary model does not build software as releases i.e. simultaneous construction of similar product in different stages of development by varying team; in other words, the evolutionary model can be seen as cyclic single release of the MULTIPARL model.

It is also important to note that this model will definitely increase development cost but will drastically reduce maintenance cost. Overall, it is more economical since operational (maintenance) cost is usually twice development cost (Sommerville, 1997).

Table 1: Appropriateness table of process models to web-based application development using the Onibere's selection criteria (Ekuobase, 2004)

Small scale projects			Medium scale projects			Enterprise projects		
Process model	Rank (%)	POS.	Process model	Rank (%)	POS.	Process model	Rank (%)	POS.
MULTIPARL	75.25	1st	MULTIPARL	84.34	1st	MULTIPARL	86.84	1st
Evolutionary	75.25	1st	Spiral	79.52	2nd	Spiral	78.95	2nd
Spiral	69.31	3rd	Evolutionary	71.08	3rd	Evolutionary	69.74	3rd
Prototyping	63.37	4th	Incremental	65.06	4th	Reusable model	60.53	4th
Incremental	61.39	5th	Reusable model	61.45	5th	Incremental	59.21	5th
Reusable model	59.41	6th	Prototyping	60.24	6th	Prototyping	59.21	5th
Waterfall	55.45	7th	Waterfall	46.99	7th	Waterfall	52.63	7th

CONCLUSION

The dynamics of the software construction market and the nature of software itself explain why both the conventional (rigid) and agile software processes cannot cope with the current realities of quality software development; since each underplayed on one or more critical success factors of software development: product quality, time-to-market, innovation and documentation. Consequently, a software process model that gives sufficient attention to these success factors was desired and proposed. The proposed model-the MULTIPARL model-by Onibere's software process selection scheme will cope quite comfortably with the current realities of quality software development particularly for the development of medium to large scale software.

It is however, important that the novel software process model be put to real life use or trial in order to establish empirically the feasibility and strengths and weaknesses of the model as was the case with other models in existence today. On our part, this is already ongoing for a Web based software development project but this publication is necessary to allow for quick and wide domain/project trial of this novel model. Time is not on our side!

REFERENCES

Abrahamsson, P., J. Warsta, M.T. Siponen and J. Ronkainen J., 2003. New direction on agile methods: A comparative analysis. Proceedings of 25th International Conference on Software Engineering, IEEE Computer, pp: 244.

Alliance, A., 2001. Manifesto for Agile Software Development. http://www.agilemanifesto.org.

Alur, D., J. Crupi and D. Malks, 2003. Core J2EE patterns: Best practices and design strategies. Sun Microsystems, pp: 650.

Aoyama, M., 1998. Agile Software process and its experience. Proceedings of 20th International Conference on Software Engineering (ICSE). IEEE Computer, pp: 3-12.

Basili, V.R. and A.J. Turner, 1975. Iterative enhancement. IEEE. Trans. Software Eng., 1 (4): 390-396.

Baskerville, R., L. Levine, J. Pries-Heje and S. Slaughter, 2001. How internet software companies negotiate quality. IEEE Comput., 34 (5): 51-57.

Boehm, B., 2002. Get ready for Agile methods, with care. IEEE. Comput., 35 (2): 64-69.

Boehm, B. and R. Turner, 2005. Management Challenges to Implementing Agile Processes in Traditional Development Organisations. IEEE. Software, 22 (5): 30-39.

Cockburn, A., 2002. Agile Software Development, Addison-Wesley.

Cusumano, M. and R.W. Selby, 1997. How microsoft builds software. Commun. ACM, 40 (6): 53-61.

Cusumano, M. and D. Yoffie, 1999. Software development on internet time. IEEE Comput., 32 (10): 60-69.

Ekuobase, G.O., 2004. Scaling Process Models for Web Based Application Development, M.Sc Thesis, Department of Computer Science, University of Benin, Edo State, Nigeria.

Ekuobase, G.O., 2006. Software Creative Milestones, Proceedings of International Conference on Advances in Engineering and Technology, Entebbe-Uganda, Elsevier, pp: 848-855.

Ekuobase, G.O. and E.A. Onibere, 2007. Software Process Selection Criteria in Perspective. Int. J. Physical Sci., 2 (3): 81-89.

Fowler, M., 1997. Analysis of Patterns: Reusable Object Models, Addison Wesley.

Fowler, M., 1999. Refactoring-Improving the Design of Existing Code, Addison Wesley.

Fowler, M., 2002. Patterns of Enterprise Application Architecture, Addison Wesley.

Griss, M. and G. Pour, 2001. Accelerating development with Agent components. IEEE. Comput., 34 (5): 37-43.

Highsmith, J. and A. Cockburn, 2001. Agile software development: The business of innovation. IEEE. Comput., 34 (9): 120-122.

Onibere, E.A. and G.O. Ekuobase, 2006. Enhanced software process selection criteria. J. Inst. Maths. Comput. Sci., 17 (1): 17-32.

Paulson, J.W., G. Succi and A. Eberlain, 2004. An empirical study of open-source and closed-source software products. IEEE. Trans. Software Eng., 30 (4): 246-256.

McBreen, P., 2003. Questioning Extreme Programming, Addison-Wesley.

Miller, G.G., 2001. The characteristics of agile software processes. 39th Proc. TOOLS, IEEE Computer, pp: 365.

Scacchi, W., 2001. Process models in software engineering. Encyclopedia of Software Engineering, John-Wiley.

Snowdon, R.A., 2003. Overview of Process Modelling. www.cs.man.ac.uk/ipg/Docs/pmover.html.

Sommerville, I., 1996. Software Process Models. ACM Comput. Surveys, 28 (1): 269-271.

Sommerville, I., 1997. Software Engineering, USA: Addison Wesley.