J. Software Engineering & Applications, 2010, 3, 933-938
doi:10.4236/jsea.2010.310110 Published Online October 2010 (http://www.SciRP.org/journal/jsea)

Software Architecture and Methodology as a Tool for Efficient Software Engineering Process: A Critical Appraisal

Achimugu Philip[1], Babajide Afolabi[2], Oluwaranti Adeniran[2], Gambo Ishaya[2], Oluwagbemi Oluwatolani[1]

[1]Computer Science Department, Lead City University, Ibadan, Nigeria; [2]Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria.
Email: {check4philo, igpeni, tolapeace}@yahoo.com, {bafox, aranti}@oauife.edu.ng

Received July 10th, 2010; revised August 14th, 2010; accepted August 18th, 2010.

ABSTRACT

The foundation for any software system is its architecture. Software architecture is a view of the system that includes the system's major components, the behaviour of those components as visible to the rest of the system, and the ways in which the components interact and coordinate to achieve the overall system's goal. Every efficient software system arises as a result of sound architectural basement. This requires the use of good architecture engineering practices and methods. This paper recognizes software architecture practice as a discipline pervading all phases of software development and also presents an enhanced model for software engineering process which provides an avenue for speedy, efficient and timely delivery of software products to their intended users. The integration of software architecture into the phases of software development process in a generic software life cycle is also contained in this research report. This is to enable software engineers and system analysts to use effective software architecture practices and to employ appropriate methodology during the software engineering process.

Keywords: *Software Systems, Architecture, Software Engineering, System Life Cycle, Software Components*

1. Introduction

Many people limit the term software engineering to just computer program. In the real sense of it, it is not just the program but also the associated documentation and design principles required to make these programs operate correctly. Software products may be developed for a particular customer or for general market, so they undergo series of thoughts and ideas that account for their initial inception, development, production, operation, upkeep and usability from one generation to another [1].

Software engineering process or activities therefore can be considered as sets of activities and associated results which produce a software product. They include software specification, development, validation and evolution. Software process model represents a networked sequence of activities, objects, transformations and events that embodies strategies for accomplishing software evolution. Different process models organize these activities in different ways, in different level of details and they are best suited for different project complexities.

Software architecture and methodology practice has emerged as a crucial part of the design process and is the main focus of this paper. Software architecture encompasses the structures of large software systems. The architectural view of a system is abstract, distilling away details of implementation, algorithm, and data representation and concentrating on the behaviour and interaction of "black box" elements. Software architecture is developed as the first step toward designing a system that has a collection of desired properties [2,3] put it very nicely in this formula (Software architecture = {Elements, Forms, Rationale/Constraints}).

Software methodology on the other hand, is a pre-defined sequence of events that must be executed, followed or carried out in order to produce a well structured and robust software product that meets user's requirement and produce good scalable tendencies.

Therefore, this paper argues that software engineers who have sound knowledge of software architecture and appropriate methodology to be employed in software engineering process will be better informed and hence produce good quality software and deliver same at the appropriate time, thus avoiding breach of contract which is common amongst software engineers.

2. Software Architecture Practice

Today, software architecture practice is one sub-discipline within software engineering that is concerned with the high-level (abstract) design of the software of one or more systems. Software architecture are created, evolved, and maintained in a complex environment. The architecture business cycle [1] of **Figure 1** illustrates this. On the left hand side, the figure presents different factors that influence a software architecture through an architect. It is the responsibility of the architect to manage these factors and take care of the architecture of the system. An important factor is formed by requirements, which come from stakeholders and the developing organization. The architect also has the capacity of influencing opinions of stakeholders, refine user's requirement in a way that it captures all the activities of an organization as well as determine the technicalities of the proposed software in terms of development techniques, architectural considerations, programming language (s) to be used and the extent of scalability of the database.

2.1. What is Architectural during Software Engineering Process?

During software development, what is architectural can be determined based on what architecture is use for. The criterion for something to be architectural is this: It must be a component, or a relationship between components, or a property (of components or relationships) that needs to be externally visible in order to reason about the ability of the system to meet its quality requirements or to support decomposition of the system into independently implementable pieces. The following are some corollaries of this principle:

1) *Architecture describes what is in your system.* When you have determined your context, you have determined a boundary that describes what is in and what is out of your system (which might be someone else's subsystem). Architecture describes the part that is in.

2) *Architecture is an abstract depiction of your system.* The information in an architecture is the most abstract and yet meaningful depiction of that aspect of the system. Given the architectural specification, there should not be a need for a more abstract description. That is not to say that all aspects of architecture are abstract, nor is it to say that there is an abstraction threshold that needs to be exceeded before a piece of design information can be considered architectural.

3) *What's architectural should be critical for reasoning about critical requirements.* The architecture bridges the gap between requirements and the rest of the design. If you feel that some information is critical for reasoning about how your system will meet its requirements then it is architectural. You, as the architect, are the best judge. On the other hand, if you can eliminate some details and still compose a forceful argument through models, simulation, walk-throughs, and so on about how your architecture will satisfy key requirements then those details do not belong. However, if you put too much detail into

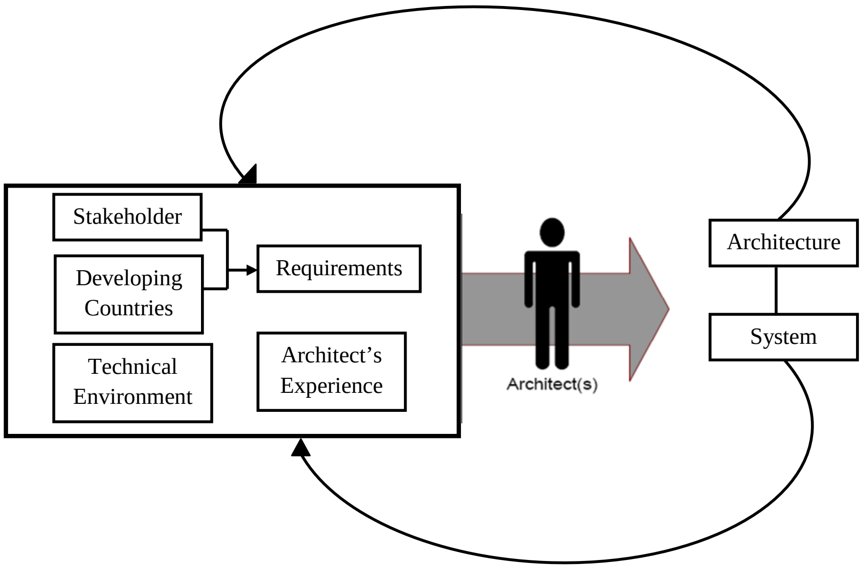


Figure 1. The architecture business cycle (Source: Bass *et al.*, 2003).

your architecture then it might not satisfy the next principle.

4) *An architectural specification needs to be graspable.* The whole point of a gross-level system depiction is that you can understand it and reason about it. Too much detail will defeat this purpose.

5) *Architecture is constraining.* It imposes requirements on all lower-level design specifications. It's good to distinguish between when a decision is made and when it is realized. For example, one can determine a process prioritization strategy, a component redundancy strategy, or a set of encapsulation rules when designing architecture; but might not actually make priority assignments, determine the algorithm for a redundant calculation, or specify the details of an interface until much later.

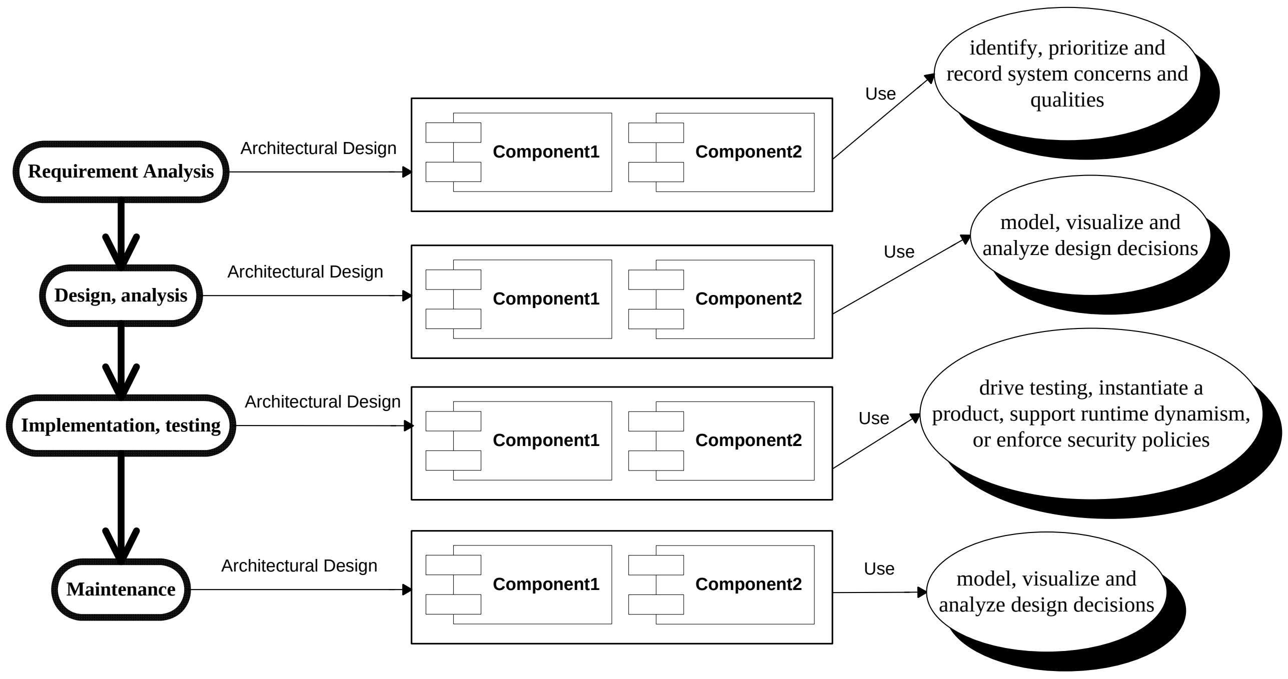
Generally, what is architectural is the most abstract depiction of the system that enables reasoning about critical requirements and constrains all subsequent refinements.

2.2. Integrating Software Architecture Practice into Software Development Process

Software architecture practice can be integrated into all the phases of software development methodologies and models [4]. This is used to distinguish it from particular analysis and design methodologies. Since the architecture determines the quality of the system, it then makes a lot of sense to have architectural design built into the software development process [5]. As shown in the **Figure 2**, software architecture is integrated into all the phases in development process. The role of software architecture in each phase of the software development process is established. The model shows that during the requirements phase of development, an architecture may be used to identify, prioritize, and record system concerns and desires. During design and analysis, an architecture may be used to model, visualize, and analyze design decisions chosen to address the principal concerns and achieve the desired qualities. Decisions may be guided by adopting one or more architectural styles. During implementation and testing, an architecture may be used to drive testing, instantiate a product, support runtime dynamism, or enforce security policies. Rather than throwing out an architecture at this point as is often done, an architecture remains part of the product. During maintenance, an architecture may be used as a basis for incorporating new features, or increasing modelling detail.

3. Methodology in Software Engineering Process

Reference [6] defines software development methodology as the framework that is used to structure, plan and control the process of developing a software product or information systems. A wide variety of such frameworks has evolved over the years, each with its own recognized



Requirement Analysis
Design, analysis
Implementation, testing
Maintenance
Architectural Design
Component1
Component2
Use
identify, prioritize and record system concerns and qualities
model, visualize and analyze design decisions
drive testing, instantiate a product, support runtime dynamism, or enforce security policies
model, visualize and analyze design decisions

Figure 2. A model integrating architecture into software development process.

strength and weaknesses. One system development methodology is not necessarily suitable for use by all projects. Each of the available methodologies is best suited for specific kinds of projects, based on various technical, organizational, project and team considerations. The framework of a software development methodology consists of:

1) A software development philosophy with the approach or approaches of the software development process.

2) Multiple tools, models and methods, to assist in the software development process.

These frameworks are often bound to some kind of organization, which further develops, support the use, and promotes the methodology. The methodology is often documented in some kind of formal documentation. This section therefore presents recommendations of appropriate methodology to be used in the software engineering process. This work is based on the theoretical study of some existing software process models. These models were ranked based on the following features:

1) Ease of use and management
2) Support for small projects
3) Support for complex projects
4) Adequate test plan
5) Support for dynamic user requirement
6) Risk analysis
7) Early delivery of project
8) Level of requirements gathered
9) Cost effectiveness
10) Meeting user's need
11) Activity based
12) Deliverable based

Reference [7] asserts that ease of use and management implies that each phase of the development process has a specific deliverable and the documentation of this makes it easy to manage. Support for project complexities (small or complex) implies effectiveness of the model when used for different projects. How and when testing is done is of great significance in the development process of software product. It implies whether it is done at the beginning, end of development or at end of each phase. In most cases, user's requirements are dynamic, so how well a model adjust to this dynamism is important. Also [8] argued that the level of user's requirements gathered that is, detailed or scanty, at the beginning phase, plays a great deal in whether the product will adequately meet user's needs. A model that adapts well with changing user requirements tends to meet users needs better. Cost effectiveness is a relative term when used in software engineering because there is always a trade-off between cash and kind implications, so this was not used to rank the models considered but its importance was not thrown away. It worth mentioning that these rankings are based entirely on findings from books and articles as referenced. The model with the best rank for each feature considered was adopted to form this optimal model.

The model that ranks highest is finally adopted for each feature in our model. Activities that lead to the achievement of these desired features are identified and the proposed model will emphasize them. This serves as the bases of the adoption. However, **Table 1** shows the various types of models and when they are best at use.

Table 1. Re-ranking of the models with best rank.

features	waterfall	incremental	Rapid prototyping	v-shape	spiral	JAD	Object process
Ease of use and management	Best	-	-	good	-	-	Better
Support for small project	Better	-	-	Best	-	-	-
Support for complex project	-	better	good	-	Better	better	Best
Test plan	-	-	better	Best	-	good	Better
Support for dynamic requirement	-	Better	Better	-	-	good	Best
Risk analysis	-	-	-	-	Best	Better	-
Early delivery of project	-	Better	Best	-	-	-	Good
Level of requirement gathered	Better	-	-	good	Better	Better	Best
Cost effective	-	-	-	-	-	-	-
Meeting user need	-	Better	Best	-	-	good	Better
Activity based	Yes	-	-	Yes	Yes	Yes	-
Delivery based	-	Yes	Yes	-	-	-	Yes

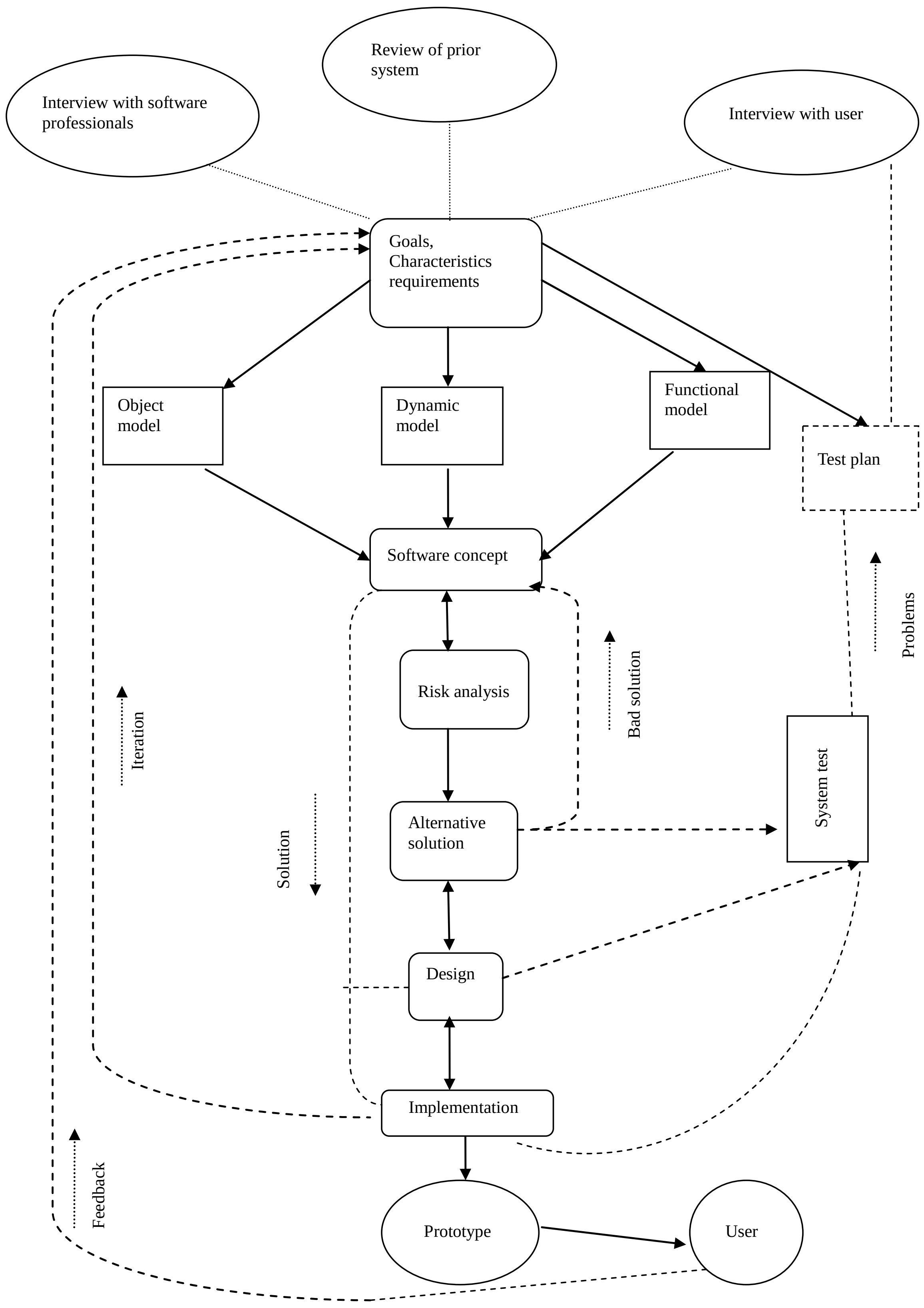


Figure 3. Enhanced model.

Figure 3 gives a pictorial description of the model. Basically the model is easy to use and manage because at the end of each phase, there is a specific deliverable and a review of the process involved. Also the phases cascade like the waterfall model but to ensure that it is not as rigid as waterfall model, the phases go back and forward that is, if there is problem in one stage, it documented and kept for next iteration where the stage will be revisited. Testing is done early before coding is done and at the end of each phase, a test plan is created to ensure quality delivery. It uses concept from object oriented analysis, design and programming to ensure support for different project complexities. Reference [9] reviewed that deliverable at the end of analysis phase is considered objects with attributes and methods; they also have constructors which are methods describing how to create the deliverable and quality assurance methods. Naturally user's requirements are dynamic, the object oriented approach of this model allows for this to be defined in a single deliverable called "task context" which can be modified without affecting the entire production process. After gathering the user requirement, a process is undertaken to identify the risk and alternate solutions. A prototype is produced at the end of this phase and this ensures that the product is delivered early to the user though with reduced functionalities. Feedback from users is used to provide a better and user oriented software. Cost effectiveness can be viewed as optimal cost for optimal solution, so this model can be said to be cost effective.

Clearly seen from the figure, requirements gathered are expanded into three views; object view represents the artefacts of the system, dynamic view represents the interaction between objects, and functional view represents methods of the system. This is the object oriented approach of the model. The phases cascade and iterate so; problems found during testing are adequately taken care of in the next iteration which corresponds to an improved version of prototype. No throwaway prototype is developed in this model because of the risk analysis which gives rise to alternate solutions.

4. Conclusions

System developers and acquirers can use effective software architecture practices across the life cycle to ensure predictable product qualities, cost, and schedule. We establish in this paper that software architecture is the bridge between mission/business goals and a software system. Secondly, software architecture drives software development throughout the life cycle, and finally the paper identifies some methodologies that could be employed during the software engineering process using some parameters. An enhanced model for software engineering process was also proposed.

REFERENCES

[1] Bass *et al.*, "Software Architecture in Practice," Addison Wesley, New York, 2003.

[2] P. Clements, "Predicting Software Quality by Architecture-Level Evaluation," *Proceedings of the Fifth International Conference on Software Quality*, Austin, Texas, October 1995.

[3] D. L. Parnas, "Designing Software for Ease of Extension and Contraction," *IEEE Transactions on Software Engineering*, Vol. 5, No. 2, 1979, pp. 128-138.

[4] B. A. Boehm, "Spiral Models of Software Development and Enhancement," *Computer*, Vol. 20, No. 9, 1987, pp. 61-72.

[5] D. Garlan, "Software Architecture: A Road Map," *Proceeding of the* 16*th International Conference on Software Engineering*, Sorrento, Italy, May 2000, pp. 71-80.

[6] M. M. Lehman, "Process Models, Process Programming, Programming Support," *Proceedings of the* 9*th International Conference on Software Engineering IEEE Computer Society*, Amsterdam, 1987, pp. 14-16.

[7] R. Lewallen, "Software Development Life Cycle Models," *Net Development*, Vol. 20, No. 9, 2006, pp. 61-72.

[8] W. W. Royce, "Managing the Development of Large Software Systems," *Proceedings of the* 9*th International Conference Software Engineering, IEEE Computer Society*, Amsterdam, 1987, pp. 328-338.

[9] W. Scacchi, "Process Models in Software Engineering," *Encyclopaedia of Software Engineering*, 2nd Edition, John Wiley and Sons, Inc., New York, 2001.