Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Comparative Analysis of Software Development Methodologies

Dr. Ifeyinwa Angela Ajah , Dr. John Otozi Ugah
Department of Computer Science
Ebonyi State University ,Abakaliki. Nigeria

Abstract*— A methodology is a formalized approach to implementing Software Development Life Cycle. There are different software development methodologies. Each one is unique because of its emphasis on process versus data and the order and focus it places on each Software Development Life Cycle (SDLC). All system methodologies lead to a representation of the system concept in terms of processes and data; however they vary in terms of whether the methodology focus primarily on business processes or on the data that support the business. Several software development methods (such as structured development, object-oriented development and rapid application development (RAD)) and associated modelling tools (such as Jackson Design and the Unified Modelling Language) have evolved to deal with issue of complexity. While these advances in methodologies and tools have helped to deal with the issue of software complexity, all these approaches share common weakness that makes them less than ideal, on their own, for the development of high integrity software. In this paper we discussed some factors which affect the approaches selected to solve problem and they include scale, quality and productivity, change, Consistency and Repeatability. Based on these factors one or a combination of the methods can be applied to develop software that will behave reliably and efficiently, affordable to develop and maintain, as well as satisfy all the requirements that customers have defined for them. Three categories of software development methodologies namely Traditional (Structured and Rapid Application Development), Agile Methodology and Object- Oriented Methodology (OOM) that attempts to balance the focus between process and data were compared. The result of this research reveals that selecting a methodology is not simple, as no one methodology is always best. For some projects a combination of methodologies will yield a quality product. The selection criteria is based on the following factors; Clarity of User Requirements, Familiarity with Technology, System Complexity, System Reliability, Short Time Schedules and Schedule Visibility. Three points scale (Bad, Good and Excellent) are used to match the suitability of the methods against the factors.

***Keywords*— OOM ,SDLC, Structured, Agile, RAD, Business process**

I. INTRODUCTION

Advances in microchip technology, the growth of internet and so on have drastically increased the demands placed upon computer. Software is used not only to provide applications on our desktop PC, or distributed business application across a network of machines, but also to control many systems around us. The growth in such embedded software, as it is known, is one of the reasons for the high rise in the demand for software in recent years [1]. Ideally all software products should be released without errors. But this is far from reality because residual errors in applications are to be expected. It is a common practice for developers of such products to release 'patches' for them. Application errors become dangerous if the consequence of its failure could result to loss of life or properties. For example errors in applications like medical control software, car break system, and air traffic control software. For these kinds of systems, the cost of software failure is dangerously high and therefore a much higher degree of confidence in the correctness of the software is required. Software engineering is concerned with addressing the challenges in Software development. It is defined as the systematic approach to the development, operation, maintenance, and retirement of software. The use of the term systematic approach for the development of software implies that methodologies are used for developing software which are repeatable. That is, if the methodologies are applied by different groups of people, similar software will be produced The key that drives software engineering are Cost, schedule and quality [2]. Cost is the cost of the resources used for the system and is dominated by the manpower cost. Schedule is the cycle time from concept to delivery and the time should be small. Quality is developing high-quality software that will satisfy user needs. Building high quality software requires that the development be broken into phases of SDLC (*planning. analysis, design, and implementation*) such that output of each phase is evaluated and reviewed so bugs can be removed. All projects require you to gather requirements, model the business needs. and create blueprints for how the system should be built; and all projects require an understanding of organizational behavior concepts like change management and team building. This is true for large and small projects; custom built and packaged: local and global. These underlying skills remain largely unchanged over time, but the actual techniques and approaches that analysts and developers use do change—often dramatically—over time [3]. Software Projects still run late and over budget, users often cannot get applications when they need them, a greater percentage of the projects are either terminated or abandoned as shown in Figure 1 , and some systems still fail to meet important user needs .

© 2013, IJARCSSE All Rights Reserved

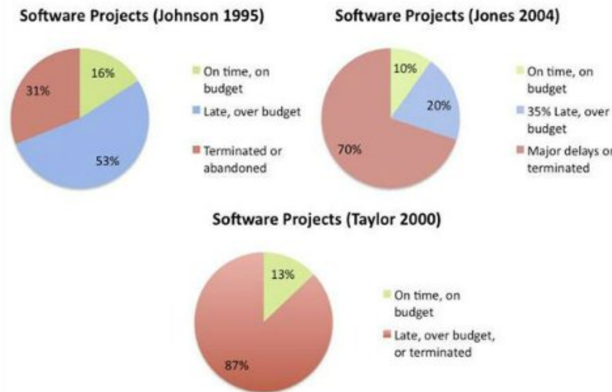

Fig 1: State of Software Development *[4]*

The basic problem therefore is to systematically develop software to satisfy the needs of some users or clients. There are some factors which affect the approaches selected to solve the problem. These factors are the primary forces that drive the progress and development in the field of software engineering and they are Scale, Quality and productivity, Consistency and Repeatability and Change.

Fig 2: Primary Challenges for Software Engineering

A.Scale

The complexity and size of software systems are continuously increasing. As the scale changes to more complex and larger software systems, new problems occur that did not exist in smaller systems (or were of minor significance), which leads to a redefining of priorities of the activities that go into developing software. Software requirements is one such area, to which little importance was attached in the early days of software development, as the emphasis was on coding and design. The tacit assumption was that the developers understood the problem clearly when it was explained to them, generally informally. As systems grew more complex, it became evident that the goals of the entire system could not be easily comprehended. Hence the need for more rigorous requirements analysis arose. Now, for large software systems, requirements analysis is perhaps the most difficult and intractable activity; it is also very error-prone. Many believe that the software engineering discipline is weakest in this critical area. A fundamental factor that software engineering must deal with is the issue of scale; development of a very large system requires a very different set of methods compared to developing a small system. In other words, the methods that are used for developing small systems generally do not scale up to large systems. An example will illustrate this point. Consider the problem of counting people in a room versus taking a census of a country. Both are essentially counting problems. But the methods used for counting people in a room (probably just go row-wise or column-wise) will just not work when taking a census. Different set of methods will have to be used for conducting a census, and the census problem will require considerably more management, organization, and validation, in addition to counting. Similarly, methods that one can use to develop programs of a few hundred lines cannot be expected to work when software of a few hundred thousand lines needs to be developed. A different set of methods must be used for developing large software. Any large project involves the use of engineering and project management. For software projects, by engineering we mean the methods, procedures, and tools that are used. In small projects, informal methods for development and management can be used. However, for large projects, both have to be much more formal, as shown in Figure 3.

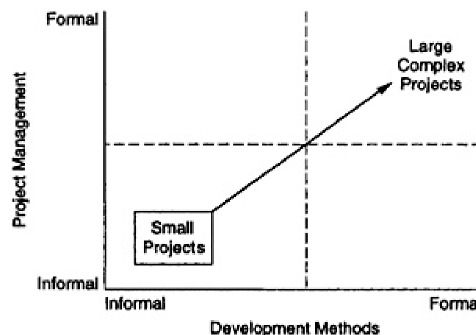

Fig 3: The problem of scale. *[2]*

© 2013, IJARCSSE All Rights Reserved

As shown in the figure, when dealing with a small software project, the engineering capability required is low (all you need to know is how to program and a bit of testing) and the project management requirement is also low. However, when the scale changes to large, to solve such problems properly, it is essential that we move in both directions—the engineering methods used for development need to be more formal, and the project management for the development project also needs to be more formal.

B. Quality and Productivity

According to the quality model adopted by international standard on software product quality, software quality comprises of six main attributes (called characteristics) [5]. These six attributes have detailed characteristics which are considered the basic ones and which can and should be measured using suitable metrics. At the top level, for a software product, these attributes can be defined as follows .

1) *Functionality:* The capability to provide functions which meet stated and implied needs when the software is used. . Functionality includes suitability (whether appropriate set of functions are provided,) accuracy (the results are accurate,) and security.Security is considered a characteristic of functionality, and is defined as "the capability to protect information and data so that unauthorized persons or systems cannot read or modify them, and authorized persons or systems are not denied access to them."
2) *Reliability:* The capability to maintain a specified level of performance.
3) *Usability:* The capability to be understood, learned, and used.
4) *Efficiency.* The capability to provide appropriate performance relative to the amount of resources used
5) *Maintainability*: The capability to be modified for purposes of making corrections, improvements, or adaptation. *Portability*: The capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product. portability has adaptability, installability etc.

C. Consistency and Repeatability

A goal of software engineering methods is that system after system can be produced with high quality and productivity. That is, the methods that are being used are repeatable across projects leading to consistency in the quality of software produced. A software development organization would like to produce consistent quality software with consistent productivity. Consistency of performance is an important factor for any organization; it allows an organization to predict the outcome of a project with reasonable accuracy, and to improve its processes to produce higher-quality products and to improve its productivity. Without consistency, even estimating cost for a project will become difficult. This requirement of consistency will force some standardized procedures to be followed for developing software. There are no globally accepted methodologies and different organizations use different ones. However, within an organization, consistency is achieved by using its chosen methodologies in a consistent manner. Frameworks like ISO9001 and the Capability Maturity Model (CMM) encourage organizations to standardize methodologies, use them consistently, and improve them based on experience.

D. Change

In today's world change in business is very rapid. It is therefore expected that software supporting businesses should respond to the changes in order to achieve the task is meant to do. Rapid change has a special impact on software. Therefore, one challenge for software engineering is to accommodate and embrace change. Different approaches are used to handle change. Approaches that can produce high quality software at high productivity but cannot accept and accommodate change are of little use today—they can solve only very few problems that are change resistant.

II. The Importance Of Specification

The complexity and size of software systems are continuously increasing. As the scale changes to more complex and larger software systems, new problems occur that did not exist in smaller systems (or were of minor significance), which leads to a redefining of priorities of the activities that go into developing software. Software requirements is one such area, to which little importance was attached in the early days of software development, as the emphasis was on coding and design. The tacit assumption was that the developers understood the problem clearly when it was explained to them, generally informally. As systems grew more complex, it became evident that the goals of the entire system could not be easily comprehended. Hence the need for more rigorous requirements analysis arose. Now, for large software systems, requirements analysis is perhaps the most difficult and intractable activity; it is also very error-prone. Many believe that the software engineering discipline is weakest in this critical area.

There are basically two reasons for a piece of software to contain error.: either the software does not conform to its specification or there are errors or omissions in the original specification. Specification therefore plays a vital role in the reliability of the software products. The design, and subsequent implementation, depends on the information in the specification, and the testing process relies upon the developers' understanding of the specification to determine whether or not the software is behaving correctly. Misunderstandings in the specification can lead to the delivery of final applications that do not match user requirements. For traditional development methods testing aims to locate these software errors. Ambiguities in the specification and the limitations of testing can result in errors in the final application. Testing is the major quality control measure used during software development. Its basic function is to detect defects in the software. During requirements analysis and design, the output is a document that is usually textual and non executable. After coding, computer programs are available that can be executed for testing purposes. This implies that

© 2013, IJARCSSE All Rights Reserved

testing not only has to uncover errors introduced during coding, but also errors introduced during the previous phases. Thus, the goal of testing is to uncover requirement, design, and coding errors in the programs.

A. Limitations of Testing

Testing cannot take place until some implementation is available

Testing can only help to uncover errors- it cannot guarantee the absence of them.

It is always carried out with respect to requirements as stated in the specification. If the specification document is in anyway ambiguous it is open to interpretation, and hence misinterpretation**.**

B. Informal vs Formal Language for Software Specification

For the majority of software applications in use today, the specification is captured in a mix of natural (informal) language and diagrams. For example the UML notation is used to specify and design systems according to the principles of object-oriented development, whereby a system is thought of as being composed of a number of fundamental units called objects. There are two important aspects to an object: the attributes (the information that it holds) and the methods or operation (the thing that it can do). Central to this is the notation of a class, which is the blueprint for all the objects belonging to that class. The figure below shows a typical UML class diagram specifying a BankAccount class.

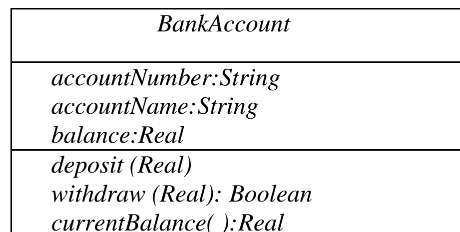

Fig 4 : A typical UML diagram for the BankAccount class

Often, a diagram such as this is supplemented by a natural language description for each method. For example, the withdraw method of the BankAccount class might have its UML specification supplemented with the following natural language description.

Withdraw: *receives a request amount to withdraw from the bank account and, if there are sufficient funds in the account, meet the request. Returns a Boolean value indicating success or failure of the attempt to withdraw money from the account.*

There is no doubt that natural language and diagrams such as this, are easy to follow by non-computing experts and so provides a good medium for discussions with clients. Unfortunately, they do not have fixed meaning (fixed semantics) from one person to the next and so are open to many different interpretations. For example considering the restrictions placed on the method that the requested amount should be withdrawn only '…if there are sufficient fund….' What is meant by the term 'sufficient'? Is it that the bank account must contain at least the amount of money that is requested for withdrawal? Or is there a minimum balance that must be maintained? Or is there an agreed overdraft limit? A Boolean value is returned from this method to indicate success or failure: does a value false indicate that an error has occurred or that there was no error? Also the amount to be withdrawn is specified to be a real number; is this to be a positive or a negative real number? All of the issues highlighted will obviously be crucial to the correct functioning of this method. This method is also incomplete and could be inconsistent with the specification of the rest of the class. A specification can be considered incomplete when the behaviour is not completely defined. In this case, the specification of the withdraw method describes what should happen when there are 'sufficient' funds in the account, but does not make clear what should happen when there are insufficient funds. Should the method withdraw as much money as is allowed or withdraw no money at all? The danger here is that the incompleteness is overlooked and that assumptions are made during design and programming, leading to the delivery of a faulty system. A specification is inconsistence when it contains contradiction. For example, an overdraft facility might be specified elsewhere. One interpretation of the withdraw method is that without funds in the bank account a given amount cannot be withdrawn. Both behaviours cannot be satisfied in an implementation.

All these amount to the conclusion that the use of these notations alone to describe critical software is not wise. To overcome these difficulties, it is desirable to use a specification notation with a fixed, unambiguous, semantics. Such notations are known as formal notations, or formal languages. Here a fixed semantics is achieved by defining a language in a completely unambiguous way using mathematical framework.

III. System Development Methodologies

All system methodologies lead to a representation of the system concept in terms of processes and data. However they vary in terms of whether the methodology focus primarily on business processes or on the data that support the business. Categorizing System Development Methodologies in terms of emphasis on process versus data we have;

© 2013, IJARCSSE All Rights Reserved

Process-Centered Methodologies

This focus first on defining the activities(the processes) associated with the system. It uses the process models as the core of the system concept. Here analysts focus first on representing the system concept as a set of processes (e.g., in Figure 5 creditworthiness score and customer's history & credit data flow in to the Compute credit risk process, and credit risk results are produced as output).

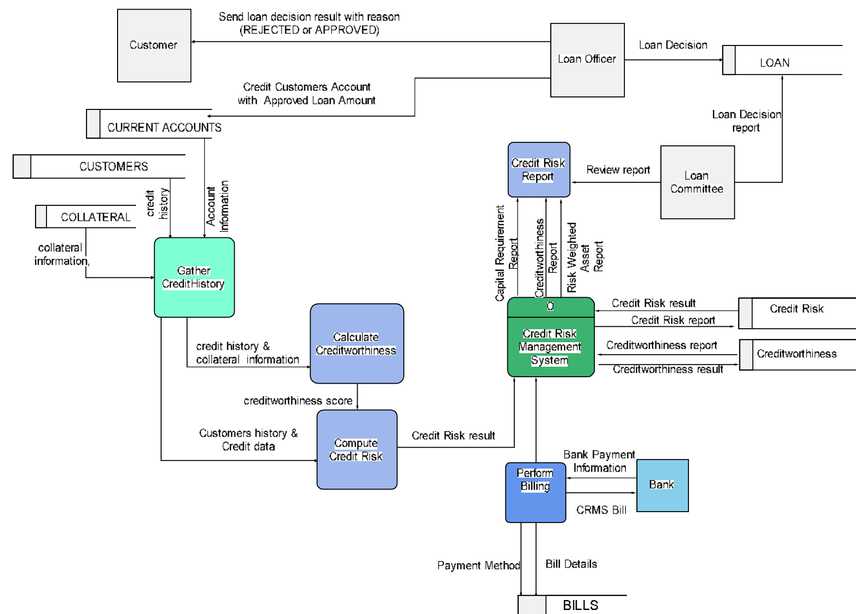

Fig 5: A Simple Model of Bank Credit Risk Management System

Data-Centred Methodologies

This focus first on defining the contents of the data storage containers and how the contents are organized. It uses data models as the core of the system concept. For example, analysts concentrates initially on identifying the data that must be available to compute credit risk and organizing them into well-defined structures(e.g., customer's national reference number, customer's bank, branch, creditworthiness score, exposure, value of collateral, guaranteed amount, etc)

D. Object –Oriented Methodologies (OOM)

Object-Oriented concept was made to balance the emphasis between process and data. It utilizes the Unified Modelling Language (UML) to describe the system concept a collection of self-contained objects, including both data and processes. Some UML tools used are ;

1) *Use case diagram:* This is used to note the type of users (Actors) of a system and what each type of user does with that system
2) *Activity diagra*m: This shows the work flow of the system; that is, it shows the flow of control from activity to activity in the system.
3) *Sequence diagram :* This shows the explicit sequence of messages that are passed between objects in the defined interaction. They are helpful for understanding real-time specifications and complex use cases.
4) *Class diagram:* This is used to define and create specific instances or object. It reflects the classes and relationships that are needed for the set of use cases which describes the system.
5) *Behavioural State diagram:* This examines the behaviour of one class. In order words it shows the different states that a single instance of a class passes through during its life in response to events, along with the responses and actions. A state is a set of values that describes an object at a specific point in time, and it represents a point in an object's life in which it satisfies some condition, performs some action, or waits for something to happen. *Package diagram:* This depicts the dependencies between the packages that make up the model.
6) *Deployment diagram:* This shows the physical architecture of the system. It can also be used to show software components being deployed onto the physical architecture.

OOM is good for modelling real-world systems. This is because in modelling real world system, processes and data are so closely related that it is difficult to pick one or the other as the primary focus. Based on this lack of congruence with the real world, new OOM have emerged that use the RAD- based sequence of SDLC phases but attempts to balance emphasis between process and data. The difference between OOM and traditional approaches like structured design is how a problem is decomposed. In traditional approaches, the problem decomposition is either process - centric or data – centric, while OOM represent the system concept in terms of process and data. Using Object Oriented Analysis and Design methods to develop real-time systems has the potential to produce safer, more reliable and maintainable code.

© 2013, IJARCSSE All Rights Reserved

Instead of using functional decomposition of the system, the OOA approach focuses on identifying objects and their activities. Using the object -oriented approach, system analysts model information systems by identifying a set of objects, along with their attributes and operations that manipulate the object data. Researchers in the object-oriented community assert that the OOA approach has many advantages in meeting the requirements of OOP [6].

Here are some of the benefits of the object-oriented approach:

1) *Reduced Maintenance:* The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviours may be reused and incorporated into new behaviours.
2) *Real-World Modelling:* Object-oriented system tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and objects are associated with behaviours. The model is based on objects, rather than on data and processing.
3) *Improved Reliability and Flexibility:* Object-oriented system promise to be far more reliable than traditional systems, primarily because new behaviours can be "built" from existing objects. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviours may be inherited from super-classes, and novel behaviours may be added without effecting existing systems functions.
4) *High Code Reusability:* When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviours from all superclasses in which it participates. When a user creates a new type of a widget, the new object behaves "wigitty", while having new behaviours which are defined to the system.

D. Categories of the System Development Methodologies

Three major categories of System Development Methodologies in terms of the progression through the SDLC phases and emphasis placed on each phase are;

Structured design

Structured design methodologies adopt a formal step-by-step approach to the SDLC that moves logically from one phase to the next. This design methodology introduces the use of formal modelling or diagramming techniques to describe a system's basic business processes and follows a basic approach of two structured design categories. It includes waterfall development and parallel development.

1) *Waterfall development* : With waterfall development- based methodologies, the analysts and users proceed sequentially from one phase to the next. The two key advantages of waterfall development-based methodologies are, the system requirements are identified long before programming begins. Changes to the requirements are minimized as the project proceeds.

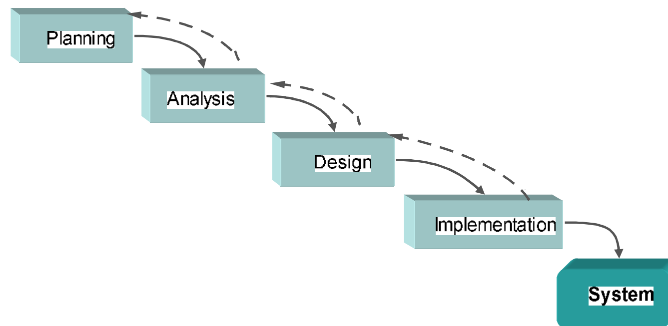

Fig 6: Waterfall Development-based Methodology. Adapted from [3]

Advantages

Simple goal, simple to understand and use.
Clearly defined stages and easy to arrange tasks.
Easy to manage. Each phase has specific deliverable and a review.
Works well for projects where requirements are well understood.
Works well when quality is more important then cost/schedule.

Disadvantages

The design must be completely specified before programming begins
Not suitable for complex projects
Not suitable for critical software
The deliverables are often a poor communication mechanism, so important requirements can be overlooked in the documentation
Not suitable for projects of long duration because in long running projects requirements are likely to change.
Users rarely are prepared for their introduction to the new system, which occur long after the initial idea for the system was introduced.

© 2013, IJARCSSE All Rights Reserved

If the project team misses important requirements, expensive post- implementation programming may be needed.
Very risky, since one process can not start before finishing the other

2) *Parallel development :* This methodology attempts to address the long time interval between the analysis phase and the delivery of the system. A general design for the entire system is performed and then the project is divided into a series of distinct subprojects that can be designed and implemented in parallel.

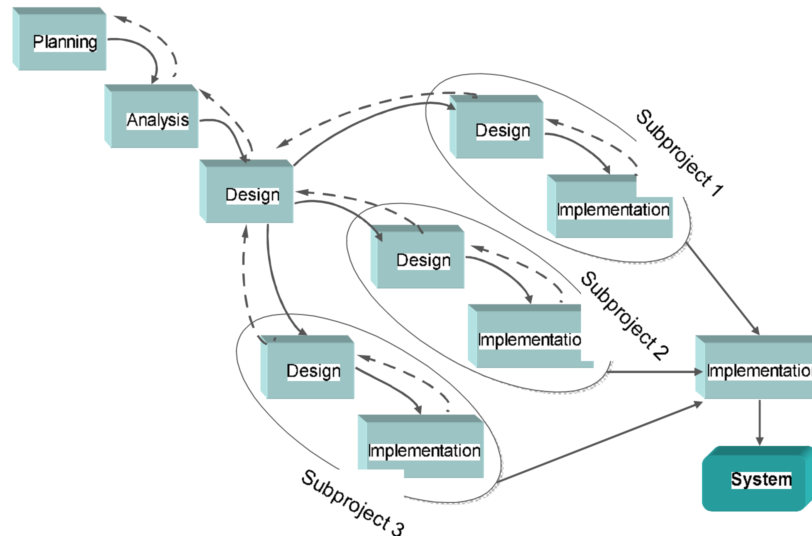

Fig 7: Parallel Development-based Methodology. Adapted from [3]

Advantages

The schedule time required to deliver a system is shortened. Thus there is less chance of changes in business environment causing rework.

Disadvantages

Just like the waterfall, it suffers from the problems caused by lengthy deliverables.

Sometimes the subprojects are not completely independent. This is because design decision made in one subprojects may affect another.

The end of the project may involve significant integration challenges.

Not suitable for complex projects as well as critical software

Rapid Application Development (RAD)

RAD-based methodologies adjust the SDLC phases to get some part of system developed quickly and into the hands of the users. Most RAD-based methodologies recommend that analysts use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as CASE (computer-aided software engineering) tools. One possible subtle problem with RAD-based methodologies is managing user expectations. It includes Phased development, prototyping and throwaway prototyping.

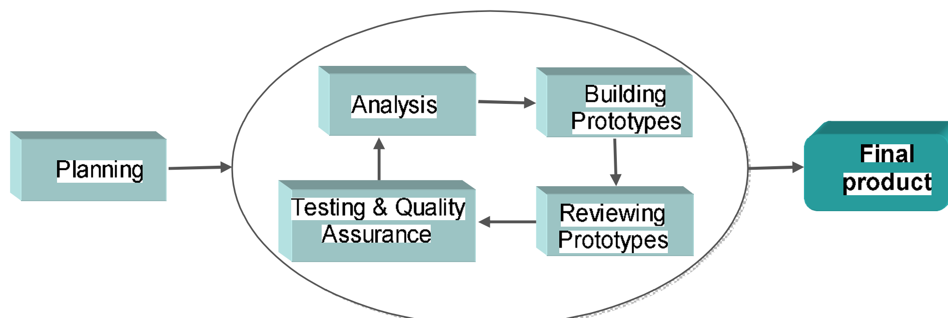

Fig 8: Rapid Application Development Methodology

1) *Phased development:* This methodology breaks the overall system into a series of versions that are developed sequentially. The team categorizes the requirements into a series of versions, then the most important and fundamental requirements are bundled into the first version of the system. The analysis phase then leads into design and implementation; however, only with the set of requirements identified for version 1. As each version is completed, the team begins work on a new version.

© 2013, IJARCSSE All Rights Reserved

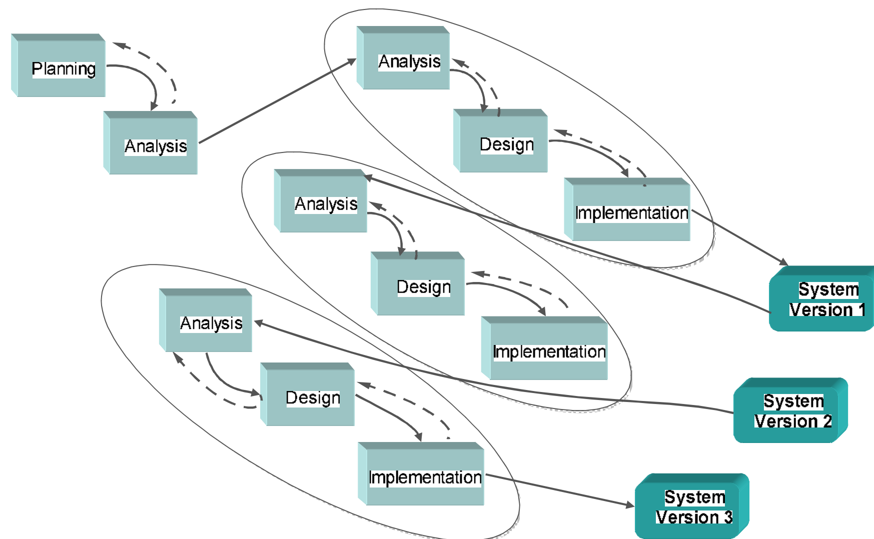

Fig 9 : Phased Development-based Methodology. Adapted from [3]

2) Prototyping: The development of the prototype typically starts when the preliminary version of the requirements specification document has been developed. Here, there is a reasonable understanding of the system and its needs and which needs are unclear or likely to change. After the prototype has been developed, the end users and clients are given an opportunity to use the prototype and play with it. Based on their experience, they provide feedback to the developers regarding the prototype: what is correct, what needs to be modified, what is missing, what is not needed, etc. Based on the feedback, the prototype is modified to incorporate some of the suggested changes that can be done easily, and then the users and the clients are again allowed to use the system. This cycle repeats until, in the judgment of the prototypers and analysts, the benefit from further changing the system and obtaining feedback is outweighed by the cost and time involved in making the changes and obtaining the feedback. Based on the feedback, the initial requirements are modified to produce the final requirements specification, which is then used to develop the production quality system.

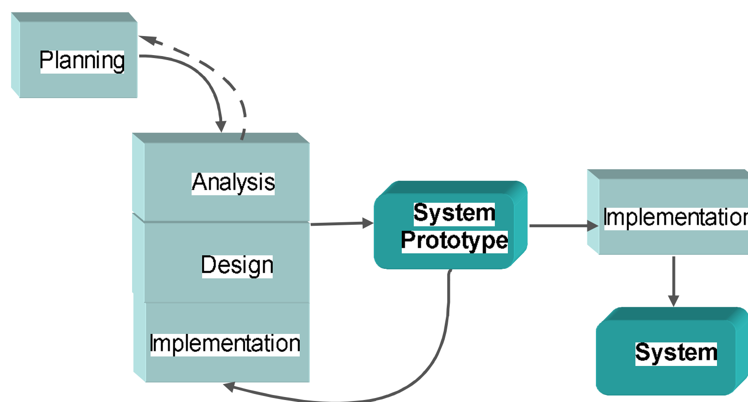

Fig 10 : Prototyping-based Methodology. Adapted from [3]

3) Throwaway prototyping: Throwaway prototyping methodologies are similar to prototyping based methodologies. The main difference is that throwaway prototyping is completed during a different point in the SDLC. It has relatively thorough analysis phase that is used to gather information and to develop ideas for the system concept. Many of the features suggested by the users may not be well understood, and handling these technical issues may be challenging. Hence, the focus of the development is to examine features that are not properly understood by analyzing, designing, and building a design prototype. The design prototype represent a part of the system that needs additional refinement, and it contains only enough details to enable users to understand the issues under consideration. Once the issues are resolved, the project moves into design and implementation. At this point, the design prototype are thrown away, which is an important difference between throwaway prototyping and prototyping, in which the prototypes evolve into the final system . This approach produces more stable and reliable system than the prototype approach but may take longer to deliver the final system.

© 2013, IJARCSSE All Rights Reserved

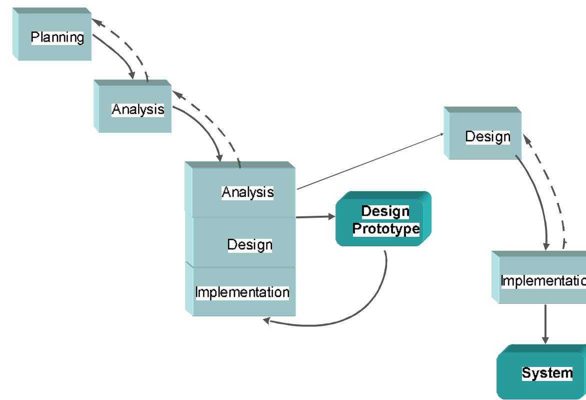

Fig 11: Throwaway Prototyping-based Methodology. Adapted from [3]

Advantages

Time to deliver is less.
Changing requirements can be accommodated.
Progress can be measured.
Cycle time can be short with use of powerful RAD tools.
Productivity with fewer people in short time.
Use of tools and frameworks.

Disadvantages

Management complexity is more.
Suitable for systems that are component based and scalable.
Requires user involvement throughout the life cycle.
Suitable for project requiring shorter development times.
Requires highly skilled developers/designers.
High dependency on modelling skills.
Inapplicable to cheaper projects as cost of modelling and automated code generation is very high for cheaper budgeted projects to befit.

Agile Development

This category focuses on streamlining the SDLC by eliminating much of the modelling and documentation overhead and the time spent on those tasks. Projects emphasize simple, iterative application development. This category uses extreme programming, which is described next.

1) Extreme Programming (XP) : The Key principles of XP include continuous testing, Simple coding and close interaction with the end users to build systems very quickly. After superficial planning process, project team perform analysis, design, and implementation phases iteratively as shown in Figure 12

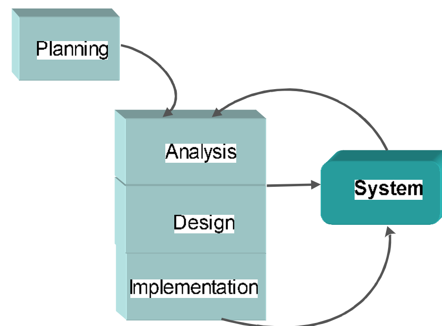

Figure 12 : An Extreme Programming-based Methodology. Adapted from [3]

Advantages

Promotes teamwork and cross training.
Functionality can be developed rapidly and demonstrated.
Resource requirements are minimum.
Suitable for fixed or changing requirements
Delivers early partial working solutions.
Good model for environments that change steadily.

© 2013, IJARCSSE All Rights Reserved

Minimal rules, documentation easily employed.
Enables concurrent development and delivery within an overall planned context.

Disadvantages

Not suitable for handling complex dependencies.
More risk of sustainability, maintainability and extensibility.
An overall plan, an agile leader and agile Project management practice is a must without which it will not work.

IV. Comparision Of Development Methodology

Selecting a methodology is not simple, as no one methodology is always best. Many organizations have their own standards. Each methodology is suitable for some context, and the main reason for studying different methodologies is to develop the ability to choose the proper model for a given project. Using a methodology as the basis, the actual process for the project can be decided, which hopefully is the optimal process for the project. The choice of a methodology is influenced by several factors: Clarity of the user requirements; familiarity with the base technology; system complexity; need for system reliability; time pressure; and need to see progress on the time schedule [3]. Table 1 summarizes some important methodology selection criteria.

TABLE 1
CRITERIA FOR SELECTING A METHODOLOGY

	Structured Methodologies		**RAD Methodologies**			**Agile Methodologies**	**Object Oriented Methodologies**
To Develop Systems with	**Waterfall**	**Parallel**	**Phased**	**Prototyping**	**Throwaway prototyping**	**XP**	**Object Oriented**
Unclear User Requirement	Bad	Bad	Good	Excellent	Excellent	Excellent	Excellent
Unfamiliar Technology	Bad	Bad	Good	Bad	Excellent	Bad	Good
Complex Systems	Good	Good	Good	Bad	Excellent	Bad	Excellent
Reliable	Good	Good	Good	Bad	Excellent	Good	Excellent
Short Time Schedule	Bad	Good	Excellent	Excellent	Good	Excellent	Excellent
Schedule Visibility	Bad	Bad	Excellent	Excellent	Good	Good	Good

A. *Selecting the Appropriate Development Methodology*

1) *Clarity of User Requirements:* Use Case driven concept of the OOM allows users and analyst to focus on how a user will interact with the system to perform single activity. This promotes better understanding and gathering of user needs. RAD methodologies of prototyping and throwaway are usually more appropriate when user requirements are unclear as they provide prototypes for users to interact with early in the SDLC.
2) *Familiarity with Technology:* If system is designed without some familiarity with the base technology, risks increase because the tools may not be capable of doing what is needed. Waterfall, Parallel development, Prototyping and Extreme Programming are appropriate when the analyst is familiar with the technology because they do not create opportunities to investigate the technology in some dept before the design is complete. Phased development does and so is good when technology is unfamiliar to the analyst. Throwaway prototyping explicitly encourages the developers to create design prototypes for areas with high risk and therefore is very suitable when technology is unfamiliar to the developers.
3) *System Complexity:* Complex systems require careful and detailed analysis and design and this is the main focus of OOM. Object Oriented approach tackles the issue of system complexity with ease. OOM concepts like polymorphism, encapsulation, and inheritance taken together allow analysts to break a complex system into smaller, more manageable components, to work on components individually, and to more easily put the components back together to form a system. Although phased development based methodology enable users to interact with the system early in the process, [3] observed that project teams who follow phased development-based methodologies tend to devote less attention to the analysis of the complete problem domain than they might if they were using other methodologies. Prototyping does not encourage detailed analysis and design and is therefore a bad choice for

© 2013, IJARCSSE All Rights Reserved

complex system. Throwaway Prototyping supports detailed analysis and design and so can effectively handle complex system. The traditional Structured methodologies is equally good for complex system, though they lack the capacity to deliver the system or prototype on time to the users and so there are chances of key issues being overlooked.

4) *System Reliability:* System reliability is usually an important factor in system development. Throwaway prototyping-based and object oriented methodologies are most appropriate when system reliability is a high priority. This is because, their emphasis on iterative and incremental development that undergoes continuous testing through out the life of the project leads to producing high quality system that is capable of meeting users needs. Prototyping-based methodologies are generally not a good choice as they lack careful analysis and design phases.
5) *Short Time Schedule:* RAD-based methodologies are well suited for projects with short time schedules because they increase speed of development. Waterfall-based methodologies are the worst choice when time is essential as they do not allow for easy schedule changes.
6) *Schedule Visibility:* Structured design methodologies do not reveal whether a project is on schedule because design and implementation takes place at the end of the project. *RAD*-based methodologies move many of the critical design decisions earlier in the project; consequently, this helps project managers recognize and address risk factors and keep expectations high.

V. Conclusion

Software is pervasive. It not only provides applications on our desktop PC, or distributed business application across a network of machines, but also control many system around us. The way people work, the choice they make, and the discipline they chose to apply, has more impact on the success of a software project. Developing a high integrity software has been a major challenge facing software developers. Research has shown that may software projects have failed and the field of software engineering is concerned with developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, satisfy all the requirements that customers have defined for them. It is important because of the impact of large, expensive software systems and the role of software in safety-critical applications. Failure in safety-critical software could result in harm to people, property or environment. Examples include medical control software and air traffic control software. The key that drives software engineering are Cost, schedule and quality. The basic problem therefore is to systematically develop software to satisfy the needs of some users. Some factors which affect the approaches selected to solve the problem are scale, quality and productivity, Consistency and Repeatability and change. High quality and productivity is governed by using good method, using good technology and user training. However, comparing between the three approaches: traditional, agile, and object oriented, there is no clear answer as which is the best approach since they all have different advantages and disadvantages. The traditional approach is perhaps the most straightforward method for systems analysis and design, however, for even smaller projects; agile methods may be more desirable. However, if the project's goal is more heavily emphasized on project complexity, scalability and component reusability, object-oriented approach could be the best choice. This work concludes that the choice of a methodology is influenced by clarity of the user requirements; familiarity with the base technology; system complexity; need for system reliability; time pressure; and need to see progress on the time schedule. To be successful in software projects, the stake holders should critical examined the trade off between different methods and can effectively combine methods that that will help achieve the objectives the software is meant to achieve.

ACKNOWLEDGMENT

Ifeyinwa Angela is a Lecturer 1 in Computer Science department of Ebonyi State University Abakaliki. Nigeria. Her research interest is in Software Engineering, Internet Programming, Databases and Object Oriented Technologies.

I would like to appreciate Dr John Otozi Ugah for his contribution in this research paper. His research interest is in Intelligent Tutoring System and Software Engineering. He is also a Lecturer 1 in Computer Science department of Ebonyi State University Abakaliki. Nigeria

References

[1] C. Quentin, and A. Kans, (2004). *Formal software Development From VDM to Java* . Palgrave Macmillian

[2] J. Pankaj, (2005). *Software Requirements Analysis and Specification An Integrated Approach to Software Engineering,* Springer Science Business Media, Inc, Third Edition, 2005.

[3] A. Dennis, B. Wixom, and R. Roth, (2006). *System Analysis and Design John* Wiley and Sons, Inc pg 171-209.

[4] D. Patterson, and A. Fox, (2013). Engineering Software as a Service: An Agile Software Approach . *In ACM Learning Webinar*. [Online]. Available: http://event.on24.com/utilApp/MediaMetricServlet?&mode=launch&mediametricid=1043816&eventid=603738&usercd=80384874. Retrieved on 17/05/13.

[5] International Standards Organization (2001). Software engineering—product quality, part 1: Quality model. Technical Report IS09126–1.

[6] Rosson, and B. Mary, (1999). Integrating Development of Task and Object Models. *Communications of the ACM, January, 1999.*

© 2013, IJARCSSE All Rights Reserved