International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Factor Analysis of Robust Secure Software Development Model

Ayoade Oluwafisayo Babatope[1], Adetunmbi Adebayo Olusola[2]

[1]Lecturer of Computer Science Department, School of Science, College of Education, Ikere Ekiti, Ekiti State,Nigeria

[2]Doctor of Computer Science Department, School of Science, Federal University of Technology, Akure, Ondo State, Nigeria

Abstract: ***Software development has advanced to a stage where quality attributes of software and information security are playing increasingly important role. The introduction of electronic commerce, mobile commerce and banking related applications have resulted to a whole new set of requirements for information systems. Besides reliability, performance and other quality attributes, the level of system security is starting to play a major role when customers or end users are making their buying decisions. There are also some clear signs that product liability laws may start taking security aspects into consideration. In the light of this there is an obvious need for a special consideration of system security when designing software products for high security applications. In this paper, we perform factor analysis using Principal Component Analysis (PCA) on the characteristics of five software development models and classify them based on key requirements for a software development project using responses from selected software developers. The objective is to help project managers and software developers select an appropriate software development model that is most suitable to the requirements of their software projects.***

Keywords: *Software Development, Software Development Models, Development Life Cycle, Information Security, Principal Component Analysis*

1. Introduction

Software development is referred to as the activity of computer programming, which is the process of writing and maintaining the source code, whereas the broader sense of the term includes all that is involved between the conceptions of the desired software through to the final demonstration of the software [8]. Therefore, software development may include research, new development, modification, reuse, re-engineering, maintenance, or any other activities that result in software products [6]

Software development is the act of working to produce/create software. This software could be produced for a variety of purposes the three most common purposes are: (i.) to meet specific needs of a specific client/business (e.g. automation of paying machines in banking system), (ii.) to meet a perceived need of some set of potential users (e.g. commercial and open source software), (iii.) for personal use (e.g. a scientist may write software to automate a mundane task)

Software, as a product, delivers the computing potential embodied by computer hardware. It is an information transformer – producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia simulation [13]

Information security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification or destruction [1]. Information security also is explained as the tasks of guarding information that is in digital format which is manipulated by a microprocessor [18]

Information security is concerned with the confidentiality, integrity and availability (CIA) of data regardless of the form the data may take: electronic, print, or other forms [1]. The field of information security has grown and evolved significantly in recent years.

Figure 1 shows the model of integrated CIA triad with respect to physical, personal and organizational levels. As a protection for software development process in information systems which comprises of hardware, software and communications; information security can be viewed as a process of identifying and applying information security industry standards, as mechanisms of protection and prevention against software threats such as snooping or wiretapping, spoofing, tampering, repudiation, or delay (denial of service-DOS), etc, at three levels or layers viz physical, personal and organizational. Essentially, procedures or policies are implemented to tell people (developers, administrators, users and operators) how to use products to ensure software security against threats within the organizations

During the last two decades, software development has made enormous progress and supporting tools have improved significantly. Different current tools have formed a second generation of instruments to support software development. Development of a reliable software system has to be approached in a systematic way and requires use of appropriate tools and mechanisms to ensure a high level of quality for the system under development. Testing is an important phase in software development life cycle that has to be managed very well. However, there is a lack of comparisons of new techniques with software development in terms of design, efficiency, code quality, testing and time effort. Hence, secured software development needs to be given more attention.

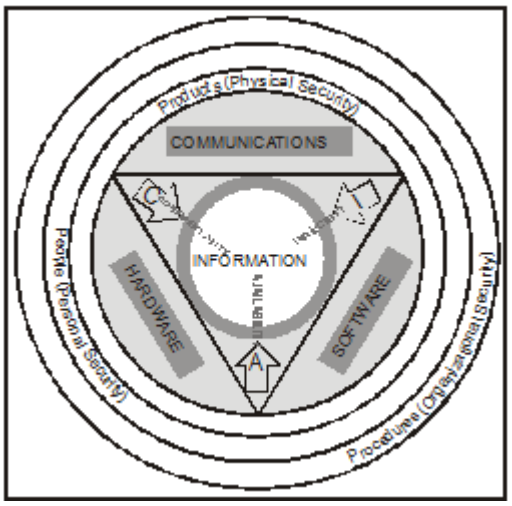


Figure 1: Model of integrated CIA triad (Layton and Timothy, 2007)

2. Literature Review

2.1 Security Polices

Security policy defines what is secure for a system; what is and what is not allowed in a system, be it an organization or a computer; or what a system is permitted to do or not to do. Security policy addresses constraints on functions and flow among them, constraints on access by external systems and adversaries including programs and access to data by people. In order to implement security policy, two methods are important. These means are security model and security mechanisms. Security model formalizes and specifies the policy while security mechanisms enforce the model and policy. Security mechanism can be anything ranging from a process to an automatic tool that enforces at least part of the security policy. A security mechanism is said to be either precise if the mechanism only allows all secure states or broad if the mechanism allows also insecure states.

2.2 Risk Analysis

Risk is defined as the combination of the probability of occurrence of harm and the severity of that harm [9]. Risk can be expressed as:

$$Risk = Threat \times Vulnerability \times Impact$$

Risk is also the expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particular harmful result [5] while Risk analysis is a process that systematically identifies valuable system resources and threats to those resources, quantifies loss exposures (i.e., loss potential) based on estimated frequencies and costs of occurrence, and (optionally) recommends how to minimize total exposure. [9] defines risk analysis in the following manner:

1) Risk is the probability of realization of a threat.
2) Risk impact is the loss caused by a realized threat.
3) Risk exposure is the expected value of a threat, that is, risk of the threat multiplied by risk impact of the same threat.
4) Risk mitigation is a course of action that can reduce the probability, loss or both associated with a certain threat.

However, risk mitigation can be divided into four different actions of avoiding risk. They are:

- actions that avoid the risk;
- actions that reduce the risk,
- actions that transfer the risk, and
- actions that assume the risk.

Nevertheless, cost-benefit analysis on choosing what risks to mitigate and how to mitigate is based on computing risk leverages given by:

$$RiskLeverage = \frac{riskexposurebeforemitigation - riskexposureaftern}{costofriskmitigation}$$

2.3 Access Control Models

Access control is a process whereby access to the resources of a system is limited to authorized users, programs, processes, or other systems [11]. It can also be defined as the ability to allow or deny access to different resources based on the accessing entity.

Access control consists of authentication – "*Who is accessing the system*?" authorization – "*Can the user do this?*" and possibly of auditing (accountability) – "*Check what the user has done*". Security policies and security models that model those policies can use two kinds of access control, either alone or in combination (Figures 2 (a & b)).

The first is known as Discretionary Access Control (DAC) where the owner of the data decides who can access the data while the other is known as Mandatory Access Control (MAC) where some higher authority controls the access and owner of the data cannot override it.

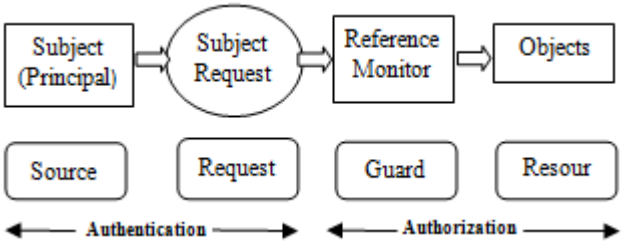


Figure 2: Diagrammatic representation of Access control [10]

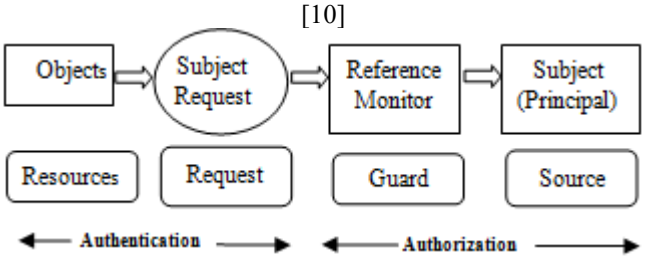


Figure 3: Diagrammatic representation of Information flow control [10]

2.4 Software Development Process

Software development process (also known as Software Life Cycle or Software process) is a structure imposed on the development of a software product. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Nevertheless, a lifecycle model is considered a more general term while a software development process a more specific term.

Software development process consists of different varieties of tasks or activities of which majority of the software products developed are mandated to follow in order to be referred to as secured software for any particular target. The various tasks/activities are as follows:

1) **Planning:** Planning is the first priority activity of creating secure software for a particular target. The most important task in creating a software product is extracting the requirements or requirements analysis. Customers typically have an abstract idea of what they want as an end result, but not what software should do. Incomplete, ambiguous, or even contradictory requirements in planning are recognized by skilled and experienced software developers/engineers at this point. Once the general requirements are gathered from the client, an analysis of the scope of the development should be determined and clearly stated often called a *scope document*.
2) **Implementation, testing and documenting**: Implementation is the part of the process where software developers actually program the code for the project. Software testing is an integral and important part of the software development process and it is a process that ensures that defects are recognized as early as possible. Documenting is the internal design of software for the purpose of future maintenance and enhancement and is done throughout software development process. It is very important to document everything in the project as this will assist in the phase of deployment and maintenance of the software.
3) **Deployment and maintenance:** Deployment is the process of implementing the code that has been written and tested for the particular target and it starts after the code is appropriately tested, approved for release and sold or otherwise distributed into a production environment. Software Training and Support is important and a lot of developers fail to realize that. Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. It may be necessary to add code that does not fit the original design to correct an unforeseen problem or it may be that a customer is requesting more functionality and code can be added to accommodate their requests. Bug Tracking System tools are often deployed at this stage of the process to allow development teams to interface with customer/field teams testing the software to identify any real or perceived issues. These software tools, both open source and commercially licensed, provide a customizable process to acquire, review, acknowledge, and respond to reported issues.

3. Software Development Models

This paper elucidate five software development models on their roles, process, responsibilities to the development & environment, practices, adoption (change property), experiences, scope of use and future plans about the software development models. The selected software development models that are considered for this analysis are as follows:

1) Waterfall Model (or classic or linear sequential model)
2) Spiral Model
3) Unified Process model
4) Scrum (Iterative or incremental) model
5) Extreme Programming – XP (Agile Modeling)

3.1 Waterfall Software Development Model

The simplest, the oldest and the most well-known software development life cycle model (SDLC) is the Waterfall model or linear sequential model or classic model, which states that the phases are organized in a linear order (Figure 3(a)). The waterfall approach emphasizes a planned and structured progression between defined phases. Each phase consists of a definite set of activities and deliverables that must be accomplished before the subsequent phase can begin. The phases are always named differently but the basic idea is that the first phase tries to capture *What* the system will do, its system and software requirements, the second phase determines *How* it will be designed. The third stage is where the developers start writing the code, the fourth phase is the *Testing* of the system and the final phase is focused on Implementation tasks such as training and heavy documentation [17].

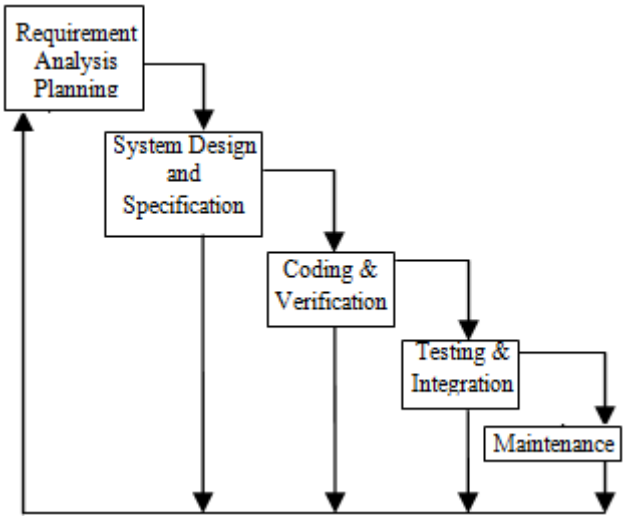


Figure 4: The Waterfall Software Life Cycle Model [17]

A project begins with feasibility analysis. On the successful demonstration of the feasibility analysis, the requirements analysis and project planning begins. The design starts after the requirements analysis is done. And coding begins after the design is done. Once the programming is completed, the code is integrated and testing is done. On successful completion of testing, the system is installed. After this the regular operation and maintenance of the system takes place.

With the waterfall model, the activities performed in a software development project are requirements analysis, project planning, system design, detailed design, coding and unit testing, system integration and testing. The waterfall model shows a process, where developers are to follow these phases in order [17]:

a) System Specification (Requirements specification & Requirements analysis): A Software Requirements Specification (SRS) is a complete description of the behavior of the system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software.

b) Software Design: Software architectural design is a process of problem-solving and planning for a software solution. After the purpose and specifications of software are determined, software developers will design or employ designers to develop a plan for a solution. It includes low-level component and algorithm implementation issues as well as the architectural view. The software requirements analysis (SRA) step of a software development process yields specifications that are used in software engineering.
c) Implementation (or Coding): Implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system. Different types of implementations may exist for a given specification or standard. For example, web browsers contain implementations of World Wide Web Consortium-recommended specifications (W3C), and software development tools contain implementations of programming languages.
d) Integration: System integration is the bringing together of the component subsystems into one system and ensuring that the subsystems function together as a system. In information technology, systems integration is the process of linking together different computing systems and software applications physically or functionally. Software integration consists of different methods which include:
   - Vertical Integration: the process of integrating subsystems according to their functionality by creating functional entities also referred to as silos.
   - Star Integration or Spaghetti Integration: is a process of integration of the systems where each system is interconnected to each of the remaining subsystems
   - Horizontal Integration or Enterprise Service Bus (ESB): is an integration method in which a specialized subsystem is dedicated to communication between other subsystems.
e) Testing (or Validation): Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing determines whether the software meets the specified requirements and finds any errors present in the code. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software.
f) Deployment (or Installation): Software deployment is all of the activities that make a software system available for use. The general deployment process consists of several interrelated activities with possible transitions between them. These activities can occur at the producer site or at the consumer site or both.
g) Maintenance: Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

3.2 Spiral Software Development Model

This is one of the recent models that have been proposed by Barry Boehm. As the name suggests, the activities in this model can be organized like a Spiral. The Spiral has many cycles. The key characteristic of this model is risk management at regular stages in the development cycle, which combines some key aspect of the waterfall model and prototyping methods, but provided emphasis in key areas which have been neglected by other methods e.g. deliberate iterative risk analysis which is particularly suited to large-scale complex systems [4].

The radial dimension represents the cumulative cost incurred in accomplishing the steps done so far and the angular dimension represents the progress made in completing each cycle of the Spiral. The structure of the Spiral model is shown in Figure 3(b). Each cycle in the Spiral begins with the identification of objectives for that cycle and the different alternatives that are possible for achieving the objectives and the imposed constraints. The Spiral model works for developed projects as well as enhancement projects.

The Spiral is visualized as a process passing through some number of iterations, with the four quadrant diagram representative of the following activities:
a) formulate plans to: identify software targets, selected to implement the program, clarify the project development restrictions;
b) Risk analysis: an analytical assessment of selected programs, to consider how to identify and eliminate risk;
c) the implementation of the project: the implementation of software development and verification;

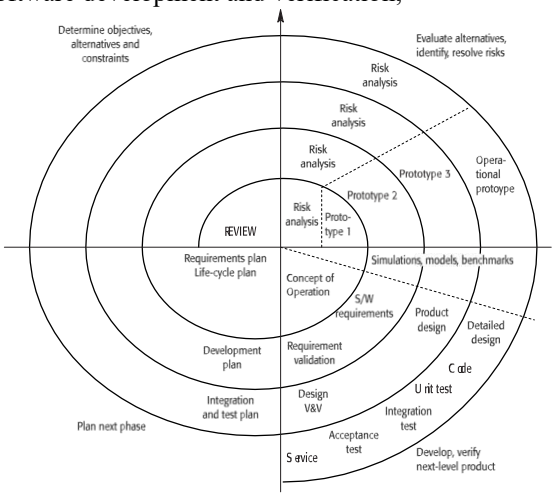


Figure 5: Spiral SDLC Model [4]

3.3 Unified Process Model

The Unified Process is a Software Engineering Process that provides a disciplined approach to assigning tasks and responsibilities within a development target. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and plan [12].

Just like other heavyweight methodologies, the Unified Process also enhances team productivity, by providing every developer with easy access to a knowledge base with guidelines, templates and tool mentors for all critical development activities (i.e. *requirement, design, test, project management, or configuration management*), so that they share a common language, process and view of how to develop software. Unified Process activities create and maintain *models* as the process that emphasizes the development and maintenance of semantically rich

representations of the software system under development. It is also a guide on how to effectively use the Unified Modeling Language (UML), an industry-standard language that allows software developers to clearly communicate requirements, architectures and designs [3]. In Unified process model, all efforts, including modeling, is organized into workflows and is performed in an iterative and incremental manner. The lifecycle of the Unified Process is presented in Figure 6.

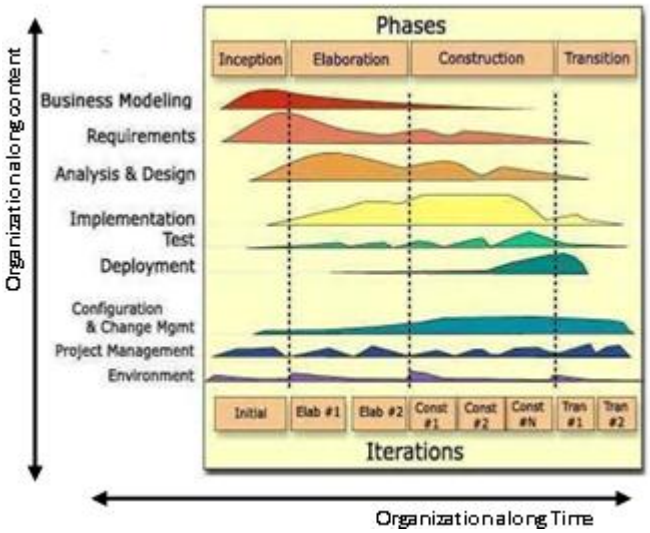


Figure 6: Unified Process Model Lifecycle [3]

The Unified Process Model is also supported by tools, which automate large parts of the process and are used to create and maintain the various models and other functions of the software engineering process i.e. visual modeling, programming, testing, etc. As a model, the Unified Process Model is a configurable process whereby no single process is suitable for all software development. It contains a Development Kit, providing support for configuring the process to suit the needs of a given target/organization.

The Unified process is subdivided into two dimensions or two axes which are the Content Dimension or Static Aspect (which is subdivided into ***9 Workflows***) and the Time Dimension or Dynamic Aspect (which is subdivided into *4 Phases*) as shown in Figure 3(c).

3.4 Scrum (Iterative) Development Model

Scrum is an iterative, incremental process for developing any product or managing system development process. It is an experimental approach that applies the ideas of industrial process control theory to system development resulting in an approach that reintroduces the ideas of flexibility, adaptability and productivity [15].

Scrum concentrates on how the team members should function in order to produce the system flexibility in a constantly changing environment which produces a potential set of functionality following expiration of iteration cycles (Figure 3(d)). The term *'scrum'* originated from a strategy in the game of rugby where it denotes "*getting an out-of-play ball back into the game*" with teamwork [15].

Scrum's main idea is that system development involves several environmental and technical variables such as requirement, time frame, resources and technology that are likely to change during the process. In the light of this, the development process is unpredictable and complex, requiring flexibility of the system development process for it to be able to respond to changes. Hence, due to the development process result, a system is produced which is useful when delivered [13].

Active management practices and tools that existed in the various phases of Scrum to avoid the problems caused by volatility and difficulty includes *Product Backlog, Sprints, Sprint Planning Meeting, Sprint Backlog, Daily Scrum, Effect Estimation* and *Sprint Meeting Review*. [16]

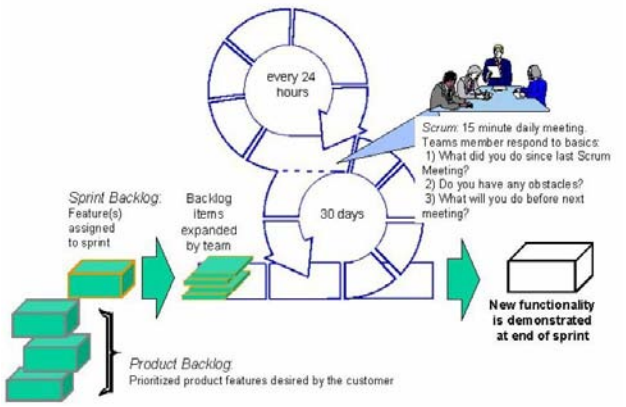


Figure 7: The Scrum process [14]

Furthermore, Scrum helps to improve the existing system engineering practices (e.g. testing practices) in an organization as it involves frequent management activities aiming at consistently identifying any deficiencies in the development process as well as the practices that are used.

3.5 Extreme Programming (XP) Model

Extreme Programming is one of the most prominent lightweight methodologies and embodies the Agile Methodology itself because it is one of the first agile processes that have been proposed. Extreme Programming, fondly referred to as XP, is an approach to software development based on the development and delivery of very small increment of functionality. XP has evolved from the problems caused by the long development cycles of traditional development models [2]

The XP process can be characterized by short development cycles, incremental planning, continuous feedback, reliance on communication, and evolutionary design [2]. It relies on constant code improvement, user involvement in the development team and pair wise programming. It can be difficult to keep the interest of customers who are involved in the process at hand whereby team members may be unsuited to the intense involvement that characterizes agile methods. In this sense, XP team members spend few minutes on programming, few minutes on project management, few minutes on design, few minutes on feedback, and few minutes on team building many times each day [19]

Hence, the term „extreme" comes from taking these commonsense principles and practices to extreme levels. Prioritizing changes can be difficult where there are multiple stakeholders (management). Maintaining simplicity requires extra work and commitment from team members and stakeholders.

The activities of the lifecycle of the XP project cycle is divided into six phases namely: Exploration, *Planning, Iteration to Release, Production, Maintenance* and *Death phases*. Figure 3(e) shows the diagram representation of the six phases of the lifecycle of the XP project system.

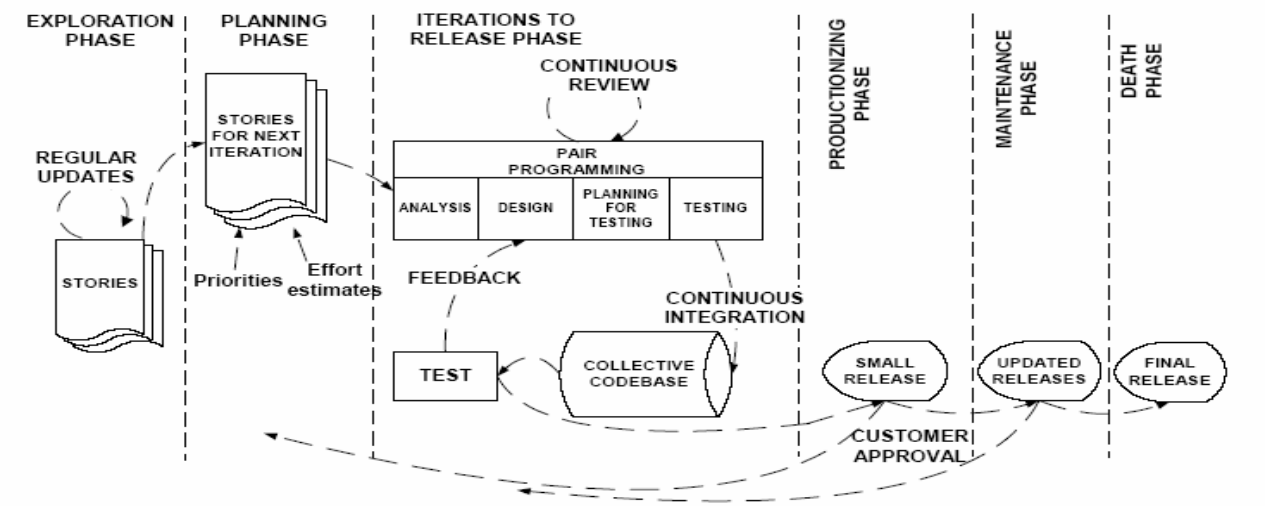


Figure 8: The Lifecycle of the XP Process [20]

However, just like in Scrum model where active actors are important to the model, the XP model also has active actors which are *Programmer, Customer, Tester, Tracker, Coach, Consultant*, and *Manager (the main boss*) [12].

One of the fundamental ideas of XP is that there is no process that fits every project but instead practices should be tailored to suit the needs of the individual projects [3]. In this sense, XP aims at enabling successful software development despite vague or constantly changing requirements in small or medium sized teams.

4. Principal Component Analysis (PCA)

Principal component analysis (PCA) is a mathematical standard tool in modern data analysis because it is a simple, non - parametric method for extracting relevant information from confusing data sets. With minimal effort PCA provides a roadmap on how to reduce a complex data set to a lower dimension to reveal the sometimes hidden, simplified structures that often underlie it [7]

Principal Component Analysis (PCA) uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called *principal components*. PCA has been used for face recognition, motion analysis and synthesis, clustering, dimension reduction, etc.

In computational terms, the principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. This process is equivalent to finding the axis system in which the co-variance matrix is diagonal. The eigenvector with the largest eigenvalue is the direction of greatest variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation and so on.

Definition: Let Σ be an n x n covariance matrix. There is an orthogonal n x n matrix ϕ whose columns are eigenvectors of Σ and a diagonal matrix γ whose diagonal elements are the eigenvalues of Σ, such that

$$\phi T \Sigma \phi = \gamma$$

Therefore, the matrix of eigenvectors ϕ as a linear transformation transforms data points in the *[X; Y]* axis system into the *[U; V]* axis system. In the general case, the linear transformation given by ϕ, transforms the data points into a data set where the variables are uncorrelated. The correlation matrix of the data in the new coordinate system is γ which has zeros in all the off diagonal elements.

The covariance matrix is used to measure how much the dimensions vary from the mean with respect to each other. In addition, the covariance of two random variables (dimensions) is their tendency to vary together is:

$$cov\ (X; Y) = E[E[X] - X] \cdot E[E[Y] - Y]$$

where$E[X]$ and $E[Y]$ denote the expected value of X and Y respectively. For a sampled dataset, this can be explicitly written out as:

$$cov\ (X;Y) = \sum_{i=1}^{N} \frac{(x_i - x)(y_i - y)}{N}$$

with$\bar{x}$ = mean(X) and $\bar{y}$ = mean(Y), where N is the dimension of the dataset. The covariance matrix, the matrix F is a matrix with elements $F_{i;j} = cov(i; j)$ centers the data by subtracting the mean of each sample vector.

With the covariance matrix, the eigenvectors and eigenvalues are calculated; where the eigenvectors are unit eigenvectors (i.e. lengths are 1). After the eigenvectors and the eigenvalues are calculated, the eigenvalues are sorted in descending order. This gives the researcher the components in order of significance. The eigenvector with the highest eigenvalue is the most dominant principal component of the

dataset (PC_1). This expresses the most significant relationship between the data dimensions. Therefore, principal components are calculated by multiplying each row of the eigenvectors with the sorted eigenvalues.

5. Methodology

The information gathered as variables were collected from the questionnaire developed for this paper which was used to identify the right methodology or software models that respondents (software developers) used to develop software for different sizes of projects.

Selected questions from the questionnaire were used as the variables. Respondents were given choices from this questions to select either 1 = Very Limited, 2 = Limited, 3 = Adequate, 4 = Extensive and 5 = Very Extensive. The variables were measured in the same units by carrying out the PCA on the original data (without standardization) which is called covariance-based PCA and the natural variance in the variables counts directly in the PCA

The number of PCs is less than or equal to the number of original variables. This transformation is defined in such a way that the first PC has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it be orthogonal to (i.e., uncorrelated with) the preceding components.

This method permits the original set of p=10 variables to be transformed to a new set of uncorrelated variables called PCs as mentioned above. These new variables are linear combinations of the original variables and each takes the form:

$$Z_j = a_{1j}X_1 + a_{2j}X_2 + a_{3j}X_3 + \ldots + a_{pj}X_p$$

where ($j = 1, \ldots, p$)

where $a_{1j}, a_{2j}, a_{3j}, \ldots a_{pj}$ is the jth eigenvector of the covariance matrix P and $X_1, X_2, X_3, \ldots, X_p$ are the variable values recorded for each of the N samples (Respondents organizations considered for the questionnaire).

The PCs are derived in decreasing order of importance where the first PCs accounts for the largest amount of the total variation in the original data. The variance of the jth principal component is equal to λj (the jth eigenvalue of P); therefore the percentage of the total variation accounted for by Zj is

$$Total\ Percentage\ Variation = \frac{\lambda_j}{\sum_{j=1}^{p} \lambda j} \times 100\%$$

For each of the principal components, Zj, we have a score calculated for each of the N respondent organizations considered by the researcher.

Multiple Regression using SPSS was used for the Modelling techniques to ascertain the continuous type responses which were used to establish combinations of variables that are good responses from respondents based on the questions which are coded in a binary type data (YES or NO), ordinal data type and continuous data type. These type of data are directly applicable to the statistical package (SPSS) employed.

Two methods were employed to identify a set of variables which were good prediction of the respondents" responses in the modelling procedures. They were:

1) Stepwise variable selection: this is conducted by entering and removing of variables from the model in a continuous way until a final model is produced. The model contains good prediction of the responses using a probability of p=0.05 for variable entry and p=0.1 for variable removal.
2) Best subsets: these do not guarantee that the single model produced is optimal but allow the researcher to look at several nearly equivalent alternative models.

6. Result

The analysis of software development process was carried out on different selected respondents out of which a sample of 10 respondents were selected using the five selected software development models based on factors as pointers for the questionnaire (i.e. process, responsibility, practices experiences and adoption, scope of use and their present research).

The corresponding tables and figures below were selected results from the large number of analyzed results for this paper work.

Table 1 and Figure 9 shows a sample of the analysis while Table 2 and Figure 10 shows the corresponding principal component (PCs) for the analysis.

Table 1: Respondents responses on Capability Standards Respondents" Rate of Software Development Model"s Usage (N = 10)

	Waterfall	Spiral	Unified Process	Scrum	XP
Very Limited	10	20	10	30	70
Limited	10	10	30	20	0
Average	40	20	40	40	20
Extensive	40	40	20	0	10
Very Extensive	0	0	0	10	0
Total	100	100	100	100	100

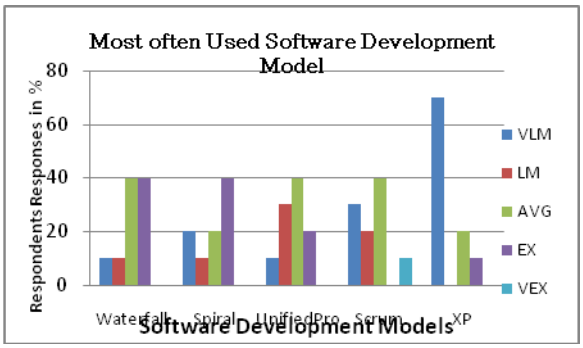


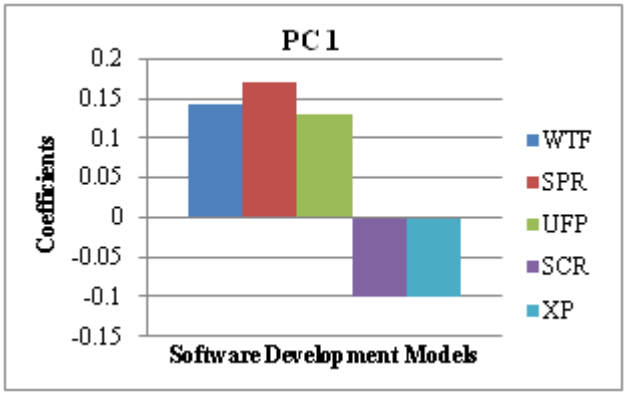
Figure 9: Respondents" responses on the most used software development model (Question 2)

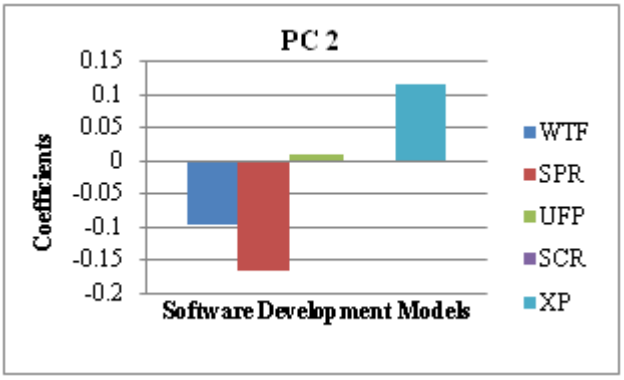
Table 2 and Figure 10 (i, ii and iii) shows the corresponding principal component (PCs) for the analysis showing the coefficients, it was observed that every software development models with large positive values for PC1, PC2

and PC3 imply those respondents that believe to use that particular software development models while those with large negative values imply otherwise.

Table 2: Respondents responses on Capability Standards

Variable	PC1	PC2	PC3	PC4	PC5
BiiWaterfall	0.141	-0.098	0.102	-0.069	0.050
BiiSpiral	0.171	-0.165	0.058	-0.013	0.112
Bii Unified Process	0.130	0.011	0.041	0.023	0.053
BiiScrum	-0.099	-0.001	0.266	0.135	-0.069
BiiXP	-0.170	0.117	0.014	-0.140	-0.036





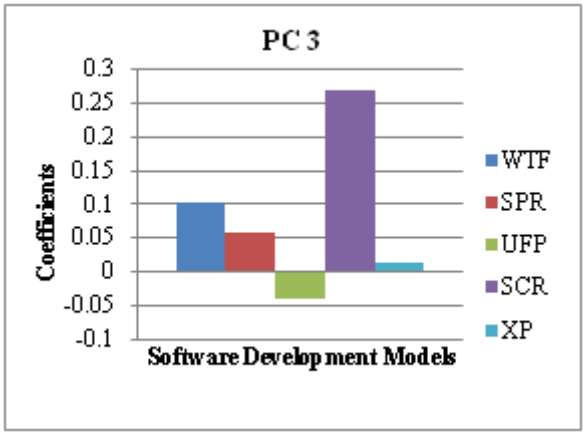


Figure 10: Principal Component on Usage

Table 3 shows Respondents" Rate of Software Development Model responses to Change and Figure 11 shows a sample of the respondents" responses for the analysis while Table 4 and Figure 12 (i, ii and iii) shows the corresponding principal component (PCs) for the analysis

Table 3: Respondents responses on Capability Standards Respondents" Rate of Software Development Model responses to Change (N = 10)

	Waterfall	Spiral	Unified Process	Scrum	XP
Very Limited	10	10	20	10	0
Limited	60	50	30	30	20
Average	10	10	10	20	10
Extensive	10	20	20	30	40
Very Extensive	10	10	20	10	30
Total	100	100	100	100	100

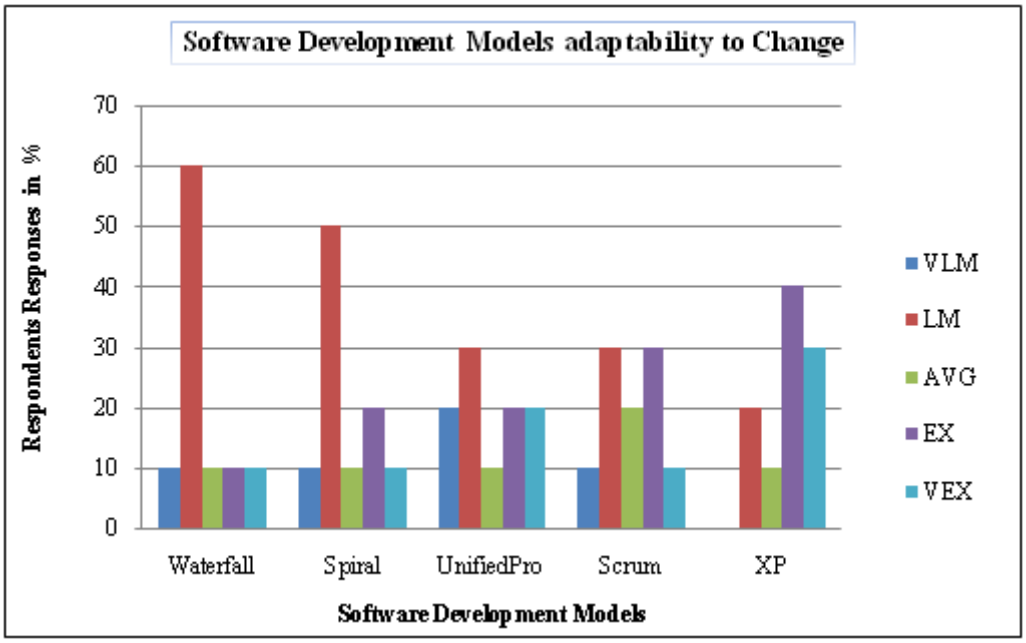


Figure 11: Respondents responses on responding to change

Table 4: Respondents responses on Capability Standards

Variable	PC1	PC2	PC3	PC4	PC5
BviWaterfall	-0.116	0.250	0.060	-0.013	0.059
BviSpiral	-0.201	0.025	-0.084	-0.056	0.100
BviUnifiedProcess	-0.247	-0.081	-0.019	-0.131	0.007
BviScrum	0.145	0.018	-0.193	-0.066	0.093
BviXP	0.141	-0.091	-0.047	0.075	-0.109

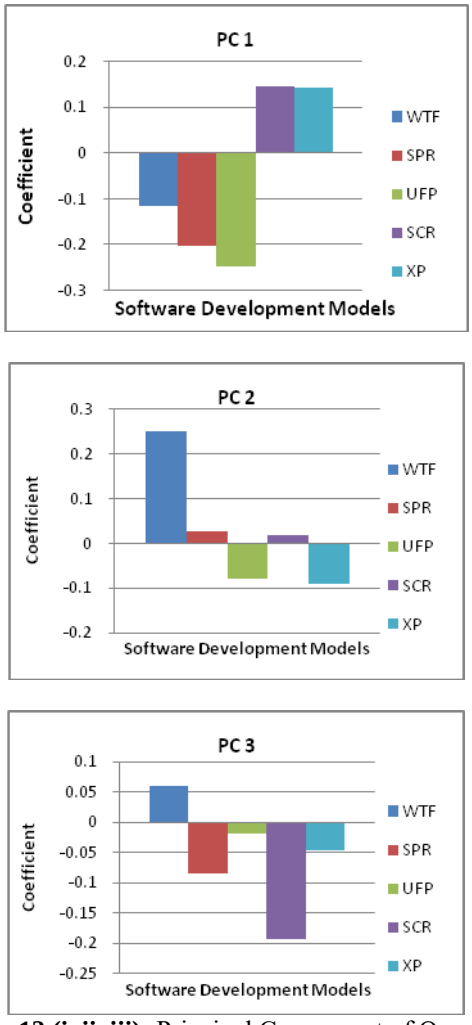


Figure 12 (i, ii, iii): Principal Component of Question 7

The above coefficients from Table 4 and corresponding Figure 12 (i, ii and iii) shows the level of software development models responding to change while developing their software with large positive values implies those respondents that believes that the model will respond to change while developing while large negative values shows otherwise. In this case, PC1 was the one that shows the proper responses that support this question of models responding to change which are the XP and Scrum models alone. The remaining PCs do not indicate this.

7. Discussion

From the data analysis based on the variables collected from respondents‟ responses and the subsequent principal component analysis (PCA) conducted on the variables, it was observed that in overall,

1) 75% of the respondents were in favor of Waterfall, Unified Process (i.e. Iterative) and XP models in software development usage.
2) Waterfall model and other heavyweight methodologies has an 80% support on management control and predefined plan structure.
3) 84% overall supported XP and Scrum models because of their support customer involvement in the team work as well as allowing changes to be done during development.
4) Respondents has an 87% support for the un-acceptance of lightweight methodologies i.e. XP and Scrum methods; for large scale developments due to their looseness of development structure.
5) Heavy documentation was always a negative aspect for the Waterfall, Spiral or Unified process models where 60% of respondents did not like the fact that heavy documentation is needed for small scale development.
6) 90% overall scale believed that developer‟s expertise and skillfulness is needed in order to ascertain the quality of the software. They believed that a larger percentage of unskilled software developers can pose more threats to the software development than the outsiders.
7) In this regard, 75% of the overall decided to use in-house software development models (i.e. RAD, FDD) that their staffs are very familiar with than employing a new one that might pose a threat to the development.
8) Hence, applying the right software development models for the right scale or level of project (i.e. small, medium or large) is of a paramount importance for any software developers as this will guide them in producing productive and suitable software for their organizations.

8. Conclusions

In conclusion, security considerations are very important factors when software needs to be developed. Threats (i.e. Denial of Service (DoS) attacks) must be identified using Threat analysis which can be powered by structured analysis approach. Attack trees and check lists can also be combined to get the best results. It is also noted that threats are usually analyzed using risk analysis method during the software module specification phases of the software development process lifecycle. The outcome points to the fact that it was really hard for security testing to find any serious weaknesses. Security testing is usually performed by developers only in functional levels while more weight was put on Quality Assurance (QA) steps which 75% of the respondents believed that it is the work of their QA department that deals with evaluation and quality assurance. In this respect the importance of specification reviews and systematic design for software development and security are very vital in software development.

Furthermore, software developers need to put more emphasis on the design of secure software system where the results are always better than to put the same amount of effort into the security testing. It was also establish that the best approach is to have a designer and a tester for software quality issues in the software system development. In spite of this, if the responsibilities are not well defined, system security engineering will not provide acceptable results based on the commitment and responsibility issues of software development and as such it would not be robust or secured.

References

[1] Allen, A.andJulia, H.(2001): The CERT Guide to SystemandNetworkSecurityPractices.Boston,MA: Addison-Wesle
[2] Beck,K.,(1999):"*EmbracingchangewithExtremePrograma mming*".IEEEComputer,Vol.32,IssueOctober.
[3] Beck,K.,(2004):"*ExtremeProgrammingexplained:Embra cechange*.Reading,Mass.,Addison-Wesley,
[4] Boehm,B.,(1998)"*ASpiralModelofSoftwareDevelopment andEnhancement*,"IEEEComputer,May199
[5] Ciampa,M.,(2007):"*Security:A guideto NetworkSecurityFundamentals*",CourseTechnology
[6] Dan,C.,(2002):*SoftwareProductManagement:Managing SoftwareDevelopmentfromIdeatoProducttoMarketingto Sales*.
[7] DepartmentofComputing(DOC),ImperialCollege,(2002): "*IntelligentDataAnalysisandProbabilisticInference*", ImperialCollege,London,Departmentof Computing
[8] DRM Associates(2002):*"NewProductDevelopmentGlossary"*.h ttp://www.npd-solutions.com/glossary.html.
[9] Fairley,R.E.(2005):,"*SoftwareRiskManagement*",IEEE Software,InstituteofElectricalandElectronicsEngineers,U SA
[10] Gasser,M.,GoldsmithA.,andKaufmanC.,(1989):*TheDigit alDistributedSystemSecurityArchitecture"*,Proc.12th NationalComputerSecurityConference
[11] Lampson,B.,Abadi,M.,Burrows,M.,andWobber,E.(1992) :"*Authenticationindistributedsystems:Theoryand practice*",ACMTrans.Comp.Systempp26-31
[12] Larman,C.,(2004):"*Agile &Iterative Development:A Manager'sGuide*".Addison-Wesley,2004.
[13] Pressman,R.S..(2005):*SoftwareEngineering:Practitioner 'sApproach*, McGraw-Hill,UnitedStatesofAmerica.
[14] Rising,L.andJanoff,N.S.,(2000): "*TheScrumsoftware developmentprocessforsmallteams*",IEEESoftware,Issue 17, pp.26-32
[15] Schwaber,K.andBeedle, M.,(2002):"*Agile Software Development with Scrum*", UpperSaddle River, NJ, Prentice–Hall,1stEdition.
[16] Schwaber,K.andBeedle,M.,(2004):"*AgileSoftwareDevel opmentwithScrum*",1stEdition,UpperSaddleRiver,NJ, Prentice–Hall.
[17] Sommerville,I..(2007):*SoftwareEngineering*,8thEdition, PearsonEducationLimited
[18] Tipton,H.F.andKrauseMickey(2000):*InformationSecurit yManagement,*4thedition,Auerbach,USA
[19] Williams,L.andCockburnA.,(2003)"*AgileSoftwareDevel opment:It'saboutFeedbackandChange*,"IEEEComputer, pp.39-43