SECURITY FRAMEWORK FOR SOFTWARE PROCESS MODELS: MEASURES FOR ESTABLISHING A CHOICE

ISSN: 2141 – 3290
www.wojast.com

[1]EGWALI A.O and [2]AKWUKWUMA V.N
Department of Computer Science.
University of Benin, P.M.B. 1154. Benin City. Nigeria.
[1]egwali.annie@yahoo.com; [2]vakwukwuma@yahoo.com

ABSTRACT

Considerable attention has been devoted to software process modeling in recent past. A well defined software process is needed to provide organizations with a framework for executing and improving activities and providing a means of reasoning about the organizational processes. The importance of incorporating Software Process Model (SPM) to system building set in motion the quest for the best criteria for selecting a suitable SPM for building software engineering projects. There has been good progress in identifying criteria for the selection of SPM, however less have been said about incorporating security into the life cycle of a system. With the different forms of cyber and identity attacks moving up the stack and into the application layer, it is becoming critical that software developers protect their customers by embedding security and privacy into their software. Security should be a factor throughout the whole life cycle of a system in order for development managers and information technology policy-makers to appraise the state of the security in development and create a vision and road map for reducing customer risk. This paper proposed a framework for selecting a suitable SPM that integrates security measures throughout a system's life cycle. The criteria for selecting a SPM was categorized into five classes: problem, product, personnel, organizational and resource. Included are factors on security measures as it affects usability, human resources, productivity, financial resources, and manageability. The framework consists of thirty-six criteria comprising six function point values each.

INTRODUCTION

According to Curtis *et al* (1988) a software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. In contrast to software life cycle models, software process models can be used to develop more precise and formalized descriptions of software life cycle activities. Their power emerges from their utilization of a sufficiently rich notation, syntax, or semantics, often suitable for computational processing (Scacchi, 2001). The importance of incorporating SPM to system building set in motion the search for the best criteria for selecting a suitable SPM for building software engineering projects. There has been good progress in identifying criteria for the selection of SPM, however less have been said about incorporating security into the life cycle of a system. Software is secured only if it can function properly despite malicious attacks (Akwukwuma and Egwali, 2008).

Security cannot be scrammed on the software at the later phases of software development life cycle, instead like other aspects of information processing systems; a security process is a continuous process that must be incorporated at each phase of the software development life cycle. Managing computer security at multiple levels brings many benefits. Each level contributes to the overall computer security program with different types of expertise, authority, and resources. In general, higher-level officials better understand the organization as a whole and have more authority. On the other hand, lower-level officials (at the computer facility and applications levels) are more familiar with the specific requirements, both technical and procedural, and problems of the systems and the users. We proposed a framework for the selection of a suitable SPM that integrates security measures throughout a system's life cycle.

SECURITY FRAMEWORK FOR SOFTWARE LIFE CYCLE MODELS

In the development of a secured system, the best approach is to draw up a security plan at the beginning of the computer system life cycle. It is a precept of the computer community that it costs ten times more to add a feature in a system after it has been designed than to include the feature in the system at the initial design phase. Basically, software life cycles include “classic” phases which are often divided into additional phases to allow better definition and control of the development process. They may also be repeated in an iterative manner, depending on the software complexity and the life cycle model used.

Requirements Phase: This phase consists of analyzing the problem for which the software is being developed; it specifies requirements, stating what the software must do. Security requirements should also be developed at the same time. These requirements can be expressed as technical features (e.g. access controls) and assurances (e.g. background checks for system developers). Security requirements can be derived from applicable standards, law, policy, and guidelines, cost-benefit trade-offs and functional needs of the system.

Functional Requirement of the System: This identifies and formalizes the objects of computation, their attributes and relationships, the operations that transform these objects and the constraints that restrict system behavior.

Specification Phase: As specifications are developed, it is necessary to undertake risk assessments. This information needs to be validated, updated, and organized into the detailed security protection requirements and specifications used by the systems designers.

Design Phase: This capability area covers practices at the requirements, architecture and design phases, including understanding and reducing product attack surfaces, threat modeling, and security design review. During this phase, the system is either built from foundation or modified. If built, security activities may include developing the system's security aspects, monitoring the development process itself for security problems, responding to changes, and monitoring threat. Threats that may arise during the development phase include Trojan horses, incorrect code, poorly functioning development tools, manipulation of code, and malicious humans. If the system is being bought, security activities may include monitoring to ensure security. In a situation where some modules of the software is built and some other modules are bought, a security analysis involving all modules will be necessary. Security choices should incorporate selection of specific off-the-shelf products, processing platform and an architecture.

Implementation Phase: This phase involves the programming of the software design with unit and integration testing being performed after software is build. Security in this phase focuses on preventing security weaknesses from being introduced, including the utilization of defensive compiler settings and code scanners. Purchased system often comes with security features disabled. These need to be enabled and configured.

Testing Phase: In this phase the software is tested for functionality and requirements compliance. Testing includes the particular parts of the system that have been developed or acquired and the testing of the entire system, this is in order to expose and remedy security weaknesses that might have been introduced at the design and implementation level.

Deployment Phase: During this phase the software is installed in the intended system and users are trained in its operation. The backing up of files, organizing training classes, managing cryptographic keys, monitoring user’s access privileges, and updating security. Changes in the system or the environment can create new vulnerabilities. In addition, system users and operators may discover new ways to intentionally or unintentionally bypass security. Strict adherence to procedures is rare over time, and procedures become outdated. By means of operational assurance the system can be reviewed to see that security controls, both automated and manual, are functioning correctly and effectively.

Maintenance Phase: This phase costs more time and effort than the original development when fixing errors and modifying or upgrading the software security.

Disposal Phase: This phase involves the disposition of information, hardware, and software. Since electronic information is easy to copy and transmit, information that is sensitive to disclosure often needs to be controlled throughout the computer system life cycle so that managers can ensure its proper disposition else these disposables will be liable to dumpster diving attacks.

Criteria for selecting the best life cycle model depend upon a number of factors. This issue was first addressed by Davis *et al* (1988), they asserted that it is difficult to compare and contrast models of software development because their proponents often use different terminology, and the models often have little in common except their beginnings (marked by a recognition that a problem exists) and ends (marked by the existence of a software solution). A framework was provided that serves as a basis for analyzing the similarities and differences among alternate life-cycle models; as a tool for software engineering researchers to help describe the probable impacts of a life-cycle mode; and as a means to help software practitioners decide on an appropriate life-cycle model to utilize on a particular project or in a particular application area.

Alexander and Davies (1991) posited that software development life cycle models can be fit into a hierarchy at different levels of abstraction. The conventional, incremental and evolutionary process models were classified at the highest level of abstraction in the hierarchy. At the next level are the waterfall, hybrid prototyping, operational specification, and transformational process models. This hierarchy divided the selection of a process model into two steps. To support the selection of a process model for a project, twenty set of criteria and three functional point values are defined. Each criteria for selecting an appropriate model was then subdivided into five categories: product, personnel, problem, organizational and resource. These selections are partially verified by looking at a set of project case studies. A deficiency in this framework is that the criteria and function point values are not sufficient to select processes for software development projects, considering the realities on ground – technological advancement, increased user requirements and new application domain like the Web (Onibere and Ekuobase, 2006).

Boehm (1989) posited that the best way to decide on a life cycle model is to use a risk driven approach based on three factors: objectives, constraints and alternatives. This was later realized in an MBASE plan and a decision table for deciding on life cycle models was proposed. Smith (2003) proposed a table that listed twenty-one project constraints that supports the selection of a process model. Boehm and Turner (2004) postulated that there are five factors involved in determining the relative suitability of agile or plain-driven methods in a particular project situation. These factors are: culture, size, criticality, personnel and dynamism. Using their framework one can tailor life cycles that range from agile to heavily plan-driven.

Little (2005) extended and simplified Boehm and Turner's idea by using more attributes in his evaluation but simplifies it by grouping them into two primary attributes; complexity and uncertainty. From an enumeration carried out on critical attributes that had influenced the success of past projects, it was discovered that the two primary attributes influenced the type of processes used.

Colouris *et al*, (2002) observed that factors such as security, technology, product mobility and technology are becoming more popular. This was affirmed by Onibere and Ekuobase (2006) who proposed a software process selection criterion that was an enhancement of what Alexander and Davies (1991) recommended. Due to the advent and heavy reliance on networked and distributed systems, they introduced seven additional factors based on security, usability, and technology and product mobility. The three point values of Alexander's framework for quantifying and measuring the selection criteria was expanded to incorporate three additional function point values in order to obtain a more robust scaling of individual process model for a given software development project.

MATERIALS AND METHOD

Ameliorated Criteria for Selecting a Software Process Model

The criteria for selecting a software process model was categorized into five classes: problem, product, personnel, organizational and resource. With each class having a six function point values each for measuring or quantifying the selection criteria. The different categories and corresponding function point values as shown in Table 1 are summarized in the rest part of this section (for more information of values definitions and applicable examples, Onibere and Ekuobase, 2006).

Proposed Framework for Selecting a Secured Software Life Cycle Models

We need criteria for selecting the right software process model irrespective of the problem domain of the system intended. Currently there is no single bullet to solving the problems and issues involved in a software development project for as time evolve, every new project brings forth new challenges that must be addressed consecutively for past mistakes not to be repeated and for superior systems to be built.

With the increase and trend of identity and malicious attacks on computer systems, security is an issue to be addressed before production commences. While it is impossible to anticipate the whole array of problems that may arise during a system's lifetime, adding new security controls to a system after a security breach can lead to chaotic security that is more expensive and less effective than an already integrated secured system.

For all systems, security should be incorporated to all the phases of the system life cycle to ensure that security keeps up with current changes due to system upgrades, system's environment and technological advancements. We need process selection criteria for deciding on an appropriate process model that will address security from the beginning to the end of the final product. We propose a framework similar to that developed by Onibere and Ekuobase (2006) with additional factors on security measures as it affects usability, human resources, productivity, financial resources, and manageability (Table 2). The framework consists of thirty-six criteria comprising six function point values each.

RESULTS AND DISCUSSION

A. Problem Criteria:

i. **Maturity of the Application**: Software development in established application area can be of great benefit to the developing team and vice versa. Values are classified as C_1 = (strange, new, familiar, standard, well-understood and master-of).

ii. **Problem Complexity:** Measures the complexity of the problem to be solved and decompose the complexity criterion into the following values: C_2 = (trivial, simple, demanding, difficult, complex and intractable).

iii. **Requirement for Partial Functionality**: Measures the practicality and or need to deliver intermediate products that provide only a part or the eventual full functionality of the target product. Values are: C_3 = (not-desired, optional, desirable, critical, urgent, and nimble.

iv. **Frequency of Change**: Estimates the frequency at which the given problem changes. Values are classified into: C_4 = (seldom, slow, moderate, fast, rapid and flashy).

v. **Magnitude of Change**: Assesses the relative size of expected changes in the problem. Applicable values are: C_5 = (insignificant, minor, small, moderate, large and extreme).

Table 1: Selection of a process model for a project based on twenty-seven set of criteria and six functional point values (Onibere and Ekuobase, 2006).

I	Criteria (Ci)	Function Point Values					
		V_{11}	V_{12}	V_{13}	V_{13}	V_{13}	V_{13}
1	User-Experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced Expert
2	User Expression Ability	Daft	Indecisive	Silent	Communicative	Expressive	Descriptive
3	Developer Experience in Application Domain	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced Expert
4	Developers Software Engineering Experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced-Expert
5	Maturity of Application Domain	Strange	New	Familiar	Standard	Well-Understood	Master Off
6	Problem Complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
7	Requirement of Partial Functionality	Not-Desired	Optional	Desirable	Critical	Urgent	Nimble
8	Requirement of Change	Seldom	Slow	Moderate	Fast	Rapid	Flashy
9	Magnitude of Change	Insignificant	Minor	Small	Moderate	Large	Extreme
10	Usability Profile	Irregular	Low-Stable	Low-High	High-Low	High-stable	High-Increase
11	Usability Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
12	Product Size	Very Small	Small	Moderate	Large	Very-Large	Extreme
13	Product Complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
14	Interface Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
15	Product Mobility Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
16	Expected Lifespan	Throwaway	Very-Short	Short	Long	Very-Long	Infinity
17	Security Requirement Performance Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
18	Performance Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
19	Funding Profile	Irregular	Low-Stable	Low-High	High-Low	High-Stable	High-Increase
20	Funds Availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
21	Staff Profile	Irregular	Low-Stable	Low-High	High-Low	High-Stable	High-Increase
22	Staff Availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
23	Access of Users	None	Restrictive	Limited	Moderate	Controlled	Free
24	Technology Requirement	Negligible	Scarce	Limited	Adequate	Ample	Abundant
25	Technology Profile	Irregular	Low-Stable	Low-High	High-Low	High-stable	High-Increase
26	Management Capability	Indifferent	Guideline	Flexible	Substantial	Enforced	Exact
27	Quality Assurance and Configuration Management Capability	Trivial	Basic	Intermediate	Substantial	Advanced	Exact

Table 2: Selection of a process model for a project based on thirty-six set of criteria and six functional point values

I	Criteria (Ci)	Function Point Values					
		F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
1	Maturity of Application Domain	Strange	New	Familiar	Standard	Well-Understood	Master Off
2	Problem Complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
3	Requirement of Partial Functionality	Not-Desired	Optional	Desirable	Critical	Urgent	Nimble
4	Frequency of Change	Seldom	Slow	Moderate	Fast	Rapid	Flashy
5	Magnitude of Change	Insignificant	Minor	Small	Moderate	Large	Extreme
6	Security of Problem Definition	Trivial	Safe	Safer	Secure	Securer	Securest
7	User-Experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced Expert
8	User Expression Ability	Daft	Indecisive	Silent	Communicative	Expressive	Descriptive
9	Developer Experience in Application Domain	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced Expert
10	Developers Software Engineering Experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced-Expert
11	System Security Cognizance	Negligible	Scarce	Limited	Adequate	Apple	Abundant
12	User Involvement and Security	Negligible	Scarce	Limited	Adequate	Apple	abundant
13	Funding Profile	Irregular	Low-Stable	Low-High	High-Low	High-Stable	High-Increase
14	Funds Availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
15	Staff Profile	Irregular	Low-Stable	Low-High	High-Low	High-Stable	High-Increase
16	Staff Availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
17	Access of Users	None	Restrictive	Limited	Moderate	Controlled	Free
18	Technology Profile	Negligible	Scarce	Limited	Adequate	Apple	Abundant
19	Technology Availability	Irregular	Low-Stable	Low-High	High-Low	High-stable	High-Increase
20	Technology Rate	Opportune	Seasonable	Batch	Timely	Well-timed	Online
21	Independent Technology Interaction	Negligible	Very-Low	Low	High	Very-High	Highest
22	Product Usability Requirement	Significant	Minor	Useful	Important	Critical	Exacting
23	Product Size	Very Small	Small	Moderate	Large	Very-Large	Extreme
24	Product Complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
25	Human Interface Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
26	Product Mobility Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
27	Expected Lifespan	Throwaway	Very-Short	Short	Long	Very-Long	Infinity
28	Security Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
29	Performance Requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
30	Usability Profile	Irregular	Low-Stable	Low-High	High-Low	High-stable	High-Increase
31	Management Capability	Indifferent	Guideline	Flexible	Substantial	Enforced	Exact
32	Quality Assurance & Configuration Management Capability	Trivial	Basic	Intermediate	Substantial	Advanced	Exact
33	Management Trust	negligible	very-low	low	High	very-high	highest
34	Management Risk	negligible	very-low	low	High	very-high	highest
35	Management Security Control	negligible	very-low	low	High	very-high	highest

We introduce a new criterion ‘Security of Problem Definition’ that addresses security concerns at the problem definition level from the commencement to the final of the software development. We define it thus:

Security of Problem Definition: Measures the level of securing the problem definition. Effective security management needs to be integrated into the management of the system from inception of the software development cycle, thus security is considered right from the problem definition stage. There should be adequate selection and implementation of appropriate technical controls and security procedures that takes care of problem definition vulnerabilities. Values are: $F_{6} =$ (trivial, safe, safer, secure, securer and securest).

B. Personnel Criteria

These criteria deal with issues relating to software developers and the users described below:

i. **Users’ Experience in Application Domain:** The users’ knowledge of the domain of the problem is appraised under this criterion. The postulated values are: C_7 = (novice, knowledgeable, experienced, well experienced, expert and experienced expert.

ii. **Users’ Ability to Express Requirement:** This criterion evaluates how well the user can communicate their needs to the developing team. Situated values are: C_8 = (daft, indecisive, silent, communicative, expressive and descriptive).

iii. **Developers’ Experience in Application Domain**: The developers’ knowledge which could result from being a user in the application domain is evaluated here using values: C_9 = (novice, knowledgeable, experienced, well experienced, expert and experienced expert).

iv. **Developers’ Software Engineering Experience**: Evaluates the developers’ experience as it relates to knowledge of the software tools, methods, techniques, technology support and languages needed for a development effort. The quantified values are: C_{10} = (novice, knowledgeable, experienced, well experienced, expert and experienced expert).

We introduce two new criteria that address security concerns as it affects personnel involved in the software development process. We define them thus:

v. **System Security Cognizance:** Measures the level of awareness and developing skills at the disposals of the system-level security personnel to develop and implement security plans that is appropriate and cost-effective. We propose the values: C_{11} = (negligible, scarce, limited, adequate, apple and abundant).

vi. **User Involvement and Security:** Measures the level of involvement of the user to the software development process. After a system's role has been defined, the security requirements implicit in that role can be defined. Security can then be explicitly stated in terms of the organization's mission. Good security practices by the user will benefit the software developing team. We propose the values: C_{12} = (negligible, scarce, limited, adequate, apple and abundant).

C. Resource Criteria

Resource criteria pertain to resources available for development discussed below:

i. **Funding Profile**: Measures the amount and availability of funds for the development effort. The values used are: C_{13} = (Irregular, Low-Stable, Low-High, High-Low, High-stable and High-Increase).

ii. **Funds Availability**: The adequacy of the funds available for an effort is measured using this criterion with the following classed value are: C_{14} = (negligible, scarce, limited, adequate, ample and abundant).

iii. **Staffing Profile:** Measures the numbers of people usable over a period of time for a software development project exercise. Values are classified into: C_{15} = (Irregular, Low-stable, Low-High, High-low, High-stable and High-increase).

iv. **Staff Availability:** Estimates the sufficiency of the available staff for a project. Applicable values are: C_{16} = (negligible, scarce, limited, adequate, ample and abundant).

v. **Accessibility of Users:** Measures the amount of access developers have to users. Values are classified as: C_{17} = (none, restrictive, limited, moderate, controlled and free).

vi. **Technology Profile**: Measures the amount and availability of technology for the development effort. The values used are: C_{18} = (Irregular, Low-stable, Low-High, High-Low, High-stale and High-Increase).

We replace this initial 'technology profile' criterion definition and introduce three new criteria that are more expansive and accommodate present development constraint as unique entities:

vii. **Technology Profile** criterion measures the amount of technological tools usable and applicable for the particular software development project to aid the software team. We introduce the values: C_{18} = (small, meager, inadequate, adequate, abundant and copious).

viii. **Technology Availability**: This criterion measures the availability of technology for the development effort. It answers the question: what quality and quantity of existing technology is at the disposal of the software development team? The values used are: C_{19} = (negligible, scarce, limited, adequate, ample and abundant).

ix. **Technology Rate**: This criterion measures the rate at which the software development team receives sound and timely information to accomplish their tasks effectively. For most organizations have trouble collecting information from myriad sources and effectively processing and distributing it within the organization. We introduce the values: C_{20} = (opportune, seasonable, batch, timely, well-timed and online).

x. **Independent Technology Interaction:** This criterion measures the level of interaction between computer security and operational elements received. In many instances, operational components obtained (i.e. modules) tend to be far larger and therefore more influential. Some software vendors seek to resolve this tension by embedding the computer security program in computer operations. This result in computer security program that lacks independence has minimal authority and as a result has few resources. We introduce the values: C_{21} = (negligible, very-low, low, high, very-high and highest).

D. Product Criteria

These criteria relates to the software product to be developed. They involve:

i. **Product Usability Requirement**: The criticality of the effortlessness with which the software can be used is measured with this criterion. It also encompasses the criticality of understanding the internal working of the system. The proposed values are: C_{22} = (insignificant, minor, useful, important, critical and exacting).

ii. **Product Size**: Measures the expected size of the final product. Since this measurement is being done at the start of the development effort, it is only an estimate. The following profile values suffice: C_{23} = (very-small, small, moderate, large, very-large and extreme).

iii. **Product Complexity**: Gauges the complexity of the software to be developed. Conditioned values are: C_{24} = (trivial, simple, demanding, difficult, complex and intractable).

iv. **Human Interface Requirement:** Measures the criticality of the human computer interface. Values are: C_{25} = (insignificant, minor, useful, important, critical and exacting).

v. **Product Mobility:** Measures the criticality of installation, portability or transportability of the final product and its content. Values employed are: C_{26} = (insignificant, minor, useful, important, critical and exacting).

vi. **Expected Life Span**: Estimates the expected life span of the final product for it is applied at beginning of the development effort. Values used are: C_{27} = (throwaway, very-short, short, long, very-long and infinite).

vii. **Product Requirement Security**: Measures the criticality of security in the final product. Values are: C_{28} = (insignificant, minor, useful, important, critical and exacting).

viii. **Product Performance Requirement:** Measures the criticality of efficiency, reliability and accuracy in the final product using values: C_{29} = (insignificant, minor, useful, important, critical and exacting).

Usability Profile is a measure applicable to the final product and so best classified as a product criterion.

ix. **Usability Profile**: This criterion measures the degree of use the resultant product will be put into. The following values suffice: C_{30} = (Irregular, Low-stable, Low-High, High-Low, High-stable and High-increase).

E. Organizational Criteria

Analysis organizational policies. The criteria are:

i. **Management Compatibility:** The degree of compatibility between an organization development requirement and the software process model is measured with this criterion, utilizing the following values: C_{31} = (indifferent, guideline, flexile, substantial, enforced and exact).

ii. **Quality Assurance and Configuration Management Capability**: This criterion measures the compatibility between a particular process model and the organization's quality assurance and configuration management procedures. Values employed are: C_{32} = (trivial, basic, intermediate, substantial, advanced and exact).

We introduce three new criteria 'Management Trust', 'Management Risk' and 'Management Security Control' that addresses security concerns at the management level from the commencement to the finale of the software development exercise. We define them thus:

iii. **Management Trust**

This criterion measures the degree of trust among the management team of a given software project. If the degree of trust is low then there is need to reassess the whole management team even before the project commences else damage can range from errors harming database integrity to supposedly trusted employees defrauding a system due to innate knowledge of the systems' architecture. Values employed are: C_{33} = (negligible, very-low, low, high, very-high and highest).

iv. **Management Risk**

This criterion measures the degree of risk the organization managers and software development team are willing to accept, taking into account the cost of security controls. Values employed are: C_{34} = (negligible, very-low, low, high, very-high and highest).

v. **Management Security Control**

This criterion appraises the degree of security provided by all participants of a software development project throughout the life cycle of a system, which includes accrediting official, data users, systems users, and system technical staff. This criterion is necessary for it trigger the construction of a security plan to ensure that security is not overlooked. Values used are: C_{35} = (negligible, very-low, low, high, very-high and highest).

CONCLUSION

The importance and usefulness of implementing employing the right software selection criteria for a given project cannot be overemphasized for the consequence of choosing an inapplicable life cycle model for a given software project can be very atrocious. There has been good progress in identifying criteria for the selection of SPM, however in an era where cyber attacks and malicious activities from insiders and outsiders of software systems and users is moving up the stack, it is becoming more critical that software developers protect their customers by embedding security and privacy into the entire software process life cycle. Security cannot be scrammed on the software at the later phases of software development life cycle, instead like other aspects of information processing systems; a security process is a continuous process that must be incorporated at each phase of the software development life cycle. We therefore proposed a framework for the selection of a suitable SPM that integrate security measures throughout a system's life cycle. we intend to implement this framework and other existing frameworks in an application software project and compare the results in order to establish inconsistency of accuracy and propose the best selection criteria for a given project definition utilizing a specific life cycle model.

REFERENCES

Akwukwuma V.N. and Egwali A. O. (2008). E-Commerce: Online Attacks and Protective Mechanisms. *Asian journal of Information Technology* 7 (9): 394- 402.

Alexander L. and Davis A. (1991). Criteria for Selecting Software Process Models. In *Proceedings of COMPSAC'91*, pp 521-528.

Boehm, B. (1989). Implementing risk management. In *Software risk management tutorial* (B. Boehm, Ed.), pp. 433-440 . IEEE Computer Society, Washington DC.

Boehm, B., and Turner, R. (2004). Balancing agility and discipline: A guide for the perplexed. *Addison-Wesley, Boston.*

Curtis, B., H. Krasner, and N. Iscoe (1988). A Field Study of the Software Design Process for Large Systems, *Communications ACM*, 31, 11, 1268-1287.

Coulouris, G., Dollimore, J., and Kindberg, T. (2001). *Distributed Systems - Concepts and Design*. Addison-Wesley, 3rd edition. Pp. 772.

Davis, A. M., Bersoff, E. H. and Comer, E. R. (1988). A strategy for comparing alternative software development life cycle models. *Software Engineering, IEEE Transactions on*, Vol. 14, No. 10. pp. 1453-1461.

Little, T. (2005). Context-adaptive agility: Managing complexity and uncertainty. *IEEE Software, 22*(3), 28-35.

Onibere, E. A. and Ekuobase, G. O. (2006). Enhanced Software Process selection Criteria. Jour. Inst. Maths and Computer Sciences (Comp. Sc. Ser.) (17) 1 pp 17-32.

Scacchi Walt (2001). Process Models in Software Engineering. Available at: *http://www.ics.uci.edu/~wscacchi/Papers/SE-Encyc/Process-Models-SE-Encyc.pdf.* accessed 1st January, 2009.

Smith L. W. (2003). Software Life Cycle. Available at: www.stsc.hill.af.mil/resources/tech_docs/gsam4/chap2.pdf. accessed 22 December, 2008.