Asian Journal of Information Technology 7 (11): 463-471, 2008
ISSN: 1682-3915

Security Framework for Software Process Models: Measures for Establishing a Choice

Annie O. Egwali and Veronica V.N. Akwukwuma
Department of Computer Science, University of Benin, P.M.B. 1154, Benin City, Nigeria

Abstract: 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 more 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 study therefore, proposed a framework for the selection of a suitable SPM that integrates security measures throughout a system's life cycle.

Key words: Software Process Model (SPM), criteria, security, life cycle model, measures, choice

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. While descriptive models describes the history of how a particular software system was developed and models that may be used as the basis for understanding and improving software development processes, or for building empirically grounded prescriptive models, a prescriptive model prescribes how a new software system should be developed and are used as guidelines or frameworks to organize and structure how software development activities should be performed and in what order.

In contrast to software life cycle models, SPM often represent a networked sequence of activities, objects, transformations and events that embody strategies for accomplishing software evolution. Such 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 Software Process Model (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 as discussed in the section that deals with related work, 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). With these attacks moving up the stack and into the application layer, it is becoming more critical that software developers protect their customers by embedding security and privacy into their software.

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. We therefore, proposed a framework for the selection of a suitable SPM similar to that proposed by Smith (2003) and Onibere and Ekuobase (2006) but which integrate security measures throughout a system's life cycle.

RELATED WORK

Davis *et al.* (1988), 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

Corresponding Author: V.N. Veronica Akwukwuma, Department of Computer Science, University of Benin, P.M.B. 1154, Benin City, Nigeria

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 model and as a means to help software practitioners decide on an appropriate life-cycle model to utilize on a particular project. A cost-benefit method of selecting among conventional (waterfall), evolutionary and incremental delivery life cycles was proposed. It essentially tried to trade off functionality and delay.

Boehm (1991) posited that the best way to decide on a life cycle model is to use a risk driven approach based on 3 factors: objectives, constraints and alternatives. This was later realized in an MBASE plan and a decision (Table 1) for deciding on life cycle models was proposed.

Alexander and Davis (1991) posited that software development life cycle models can be fitted 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 is the waterfall, hybrid prototyping, operational specification and transformational process models. This hierarchy divided the selection of a process model into 2 steps. To support the selection of a process model for a project, 20 set of criteria and 3 functional point values are defined to support the selection of a process model for a project. Each criteria for selecting an appropriate model was then subdivided into 5 categories: product, personnel, problem, organizational and resource. Under each criterion, the most appropriate process model is determined. These selections are partially verified by looking at a set of project case studies. Onibere and Ekuobase (2006) observed that the criteria and function point values are not sufficient to select processes for software development projects and that consideration should be giving to technological advancement, increased user requirements and new application domain like the Web.

Smith (2003) proposed a Table 2 that listed 21 project constraints that supports the selection of a process model. Its applicability was implemented on 4 life cycle models: the waterfall, incremental, evolutionary and spiral models. The applicability is listed for the standard definitions of the models and may not apply equally to modified models.

Boehm and Turner (2004) postulated that there are 5 critical factors involved in determining the relative suitability of agile or plain-driven methods in a particular project situation. These factors, which are: culture, size, criticality, personnel and dynamism (Table 3). 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 then simplify, it by grouping them into 2 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 2 primary attributes influenced the type of processes used. To better quantify these attributes, a scoring model was devised and plotted against each project's results on a 4-quadrant graph. They used names to represent the 4 quadrants:

- Dogs for simple projects with low uncertainty
- Colts for simple projects with high uncertainty
- Cows for complex projects with low uncertainty
- Bulls for complex projects with high uncertainty

Figure 1 summarizes each quadrant's properties. Using a matrix format, they developed an approach to help determine what process practices are barely sufficient for

Table 1: Process model decision table (MBASE Table 32)

Objectives, constraints			Alternatives			
Growth envelop	Understanding of requiements	Robustness	Available technology	Architecture understanding	Model	Example
Limited			COTS		Buy COTS	Simple inventory control
Limited			4GL transform		Transform or evolutionary development	Small business DP application
Limited	Low	Low		Low	Evolutionary prototype	Advanced pattern recognition
Limited to large	High	High		High	Waterfall	Rebuild of old system
	Low	High			Risk reduction followed by waterfall	Complex situation assessment
		High		Low		High performance avionics
Limited to medium	Low	Low-medium		High	Evolutionary development	Data exploitation
Limited to large			Large reusable components	Medium to high	Capabilities to requirements	Electronic publishing
Very large		High			Risk reduction and waterfall	Air traffic control
Medium to large	Low	Medium	Partial COTS	Low to medium	Spiral	Software support environment

Table 2: Life cycle selection matrix (Smith, 2003)

Standard definitions	Waterfall	Incremental	Evolutionary	Spiral
Requirements are known and stable	*	*		
User need are unclear/not well define			*	-
An early initial operational capability is needed		*	*	
Early functionality is needed to refine requirments for subsequent deliveries		*	*	
Sigificant risks need to be addressed		-	*	*
Must interface with other systems	-	-		-
Need to integrate new or future technology			*	*
Software is large or complex	*	*	-	-
Software is small or limited in functionality		-	-	-
Software is highly interactive with user		-	-	-
Software involves client/server function	-	-	-	*
Initial cost and schedule estimates must be followed	*	*		-
Detailed documentation necessary	*	*		-
Minimize impact on current operations	*			-
Full system must be implemented	*	-		-
Reduce the number of people required		*	*	
Project management must be simpler	*	-		
System must be responsive to user needs		-	*	*
Progress must be demonstarted early		*	*	
User feedback is needed		*	*	-
Reduce the costs of fixes and corrections		-	*	-

Method is not recommended for this constraint; *Method is recommended for this constraint; - Method is satisfactory for this constraint

Table 3: Factors for determining the relative suitability of agile or plain-driven methods (Boehm and Turner, 2004)

Factor	Agility discriminators	Plan-driven discriminators
Size	Well-matched to small products and teams Raliance on tacit knowledge limits scalability	Methods evolved to handle large products and teams Hard to tailor down to small projects
Criticality	Untested on safety-critical products Potential difficulties with simple design and lack of documentation	Methods evolved to handle highly critical
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments, but a source of potentially expensive rework for highly stable environments	Detailed plans and big design up front excellent for highly stable environment, but a source of expensive rework for highly dynamic environments
Personnel	Requires continuous presence of a critical mass of scarce Cockburn level 2 or 3 experts. Risky to use non-agile level 1B people	Needs a critical mass of scarce Cockburn level 2 and 3 experts during project definition, but can work with fewer later in the project-unless the environments is highly dynamic. Can usually accommodate some level 1B people
Culture	Thrives in a culture where people feel comfortable and empowered by having many degree of feedom (Thriving on chaos)	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear policies and procedures (Thriving on order)

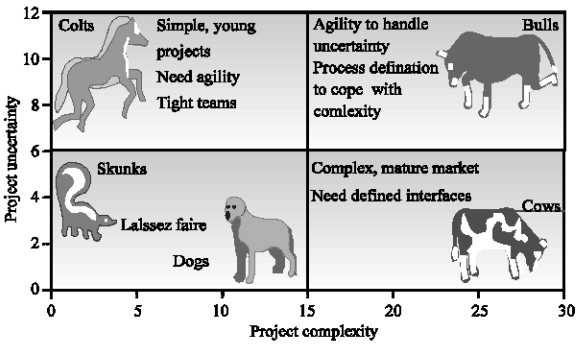

Fig. 1: Houston Matrix quadrant assessment (Little, 2005)

any given project. In his assessment, a project's structure determines its complexity and a project complexity was scored on the basis of: team size, mission criticality, team location, team maturity, domain knowledge gaps and dependencies. A project's uncertainty depends on market conditions and project constraints. The primary indicators of project uncertainty are: market uncertainty, technical uncertainty, project duration and other projects dependencies on that project and scope flexibility.

Coulouris *et al.* (2001) observed that factors such as security, technology, product mobility and technology are becoming more popular with the dawn and heavy reliance on extranet, intranet and internet systems. This was affirmed by Onibere and Ekuobase (2006), who proposed a software process selection criteria that was an enhancement of what Alexander and Davis (1991) recommended. Due to the advent and heavy reliance on networked and distributed systems, they introduced seven additional factors based on security, usability, technology and product mobility. The 3 point values of Alexander's framework for quantifying and measuring the selection criteria was expanded to incorporate three additional function point values (as we shall see in the section captioned Ameliorated Criteria for Selecting a Software Process Model), in order to obtain a more robust scaling of individual process model for a given software

development project. A deficiency in this framework is that security was addressed only in the final product and not 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. Implementing security during a system's development is less cumbersome and cost effective than implementing it later during the final product for it can interrupt continuing software operations thereby rendering all previous software production efforts futile.

Basically, software life cycles include classic phases, which are often divided into additional phases to allow better definition and control of the development process. As can be observed in the study that deals with related work done on this subject, none of the proposed criteria for selecting an appropriate life cycle model evaluated security in relation to the entire system life cycle stages. We therefore, appraise these security activities as it relates to each phase of the life cycle model.

Requirements phase: This phase consists of analyzing the problem for which the software is being developed. Security requirements should also be developed at the same time. These requirements can be expressed as technical features and assurances. 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 potentially formalizes the objects of computation, their attributes and relationships, the operations that transform these objects and the constraints that restrict system behavior. Security will support these function of the system for many aspects of the function of the system will produce related security requirements.

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. A safeguard recommended by the risk assessment could be incompatible with other requirements, or a control may be difficult to implement. Developing testing specifications early can be critical to being able to cost-effectively test security features.

Design phase: This phase involves the acquire construction of the system. 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 bought and modified. If the system is being 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. If the system is being bought, security activities may include monitoring to ensure security is a part of market surveys. In a situation, where some modules of the software are built and some other modules are bought, a security analysis involving all modules will be necessary.

Implementation phase: This phase, which involves the programming or coding of the software design is often iterative, with unit and integration testing being performed after software is build. Security in this phase focuses on preventing security weaknesses from being introduced. 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 both the testing of the particular parts of the system that have been developed or acquired and the testing of the entire system.

Deployment phase: During this phase the software is installed in the intended system and users are trained in its operation. At this point the software development effort is considered complete. However, many security activities take place during the operational phase of a system's life. 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 far more in time and effort than the original development when fixing errors and modifying or upgrading the software security to provide additional secured functionality. It is definitely much easier to change requirements earlier than it is to change software code later. This also means that software should be developed with security maintenance in mind.

Disposal phase: The disposal phase of the computer system life cycle 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. These information should either be entirely cleared making them unrecoverable by keyboard attack or completely purged by (degaussing as is the case for magnetic materials) overwriting and destruction.

AMELIORATED CRITERIA FOR SELECTING A SOFTWARE PROCESS MODEL

To address, the problem of finding the most appropriate model for a given software project when there are so many available with their unique merits and demerits in diverse domain areas, an enhanced software process selection criteria was proposed by Onibere and Ekuobase (2006). They categorized these criteria into 5 classes: problem, product, personnel, organizational and resource. With each class having a 6 function point values each for measuring the selection criteria. The different categories and their corresponding function point values are summarized in Table 4.

PROPOSED FRAMEWORK FOR SELECTING A SECURED SOFTWARE LIFE CYCLE MODEL

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 4th 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 not an issue that should be address at the end of a production process but at the very beginning even before production commences. Adding new security controls to a system after a security breach can lead to chaotic security that is definitely 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 in order to ensure that security keeps up with current changes due to system upgrades, changes in the system's environment and

Table 4: Set of criteria for selection of a process model for a project (Onibere and Ekuobase, 2006)

	Function point values					
Criteria (Ci)	V_{11}	V_{12}	V_{14}	V_{15}	V_{16}	V_{17}
User-Experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced expert
User expression ability	Daft	Indecisive	Silent	Communicative	Expressive	Descriptive
Developer experience in application domain	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced expert
Developers software engineering experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced-expert
Maturity of application domain	Strange	New	Familiar	Standard	Well-Understood	Master off
Problem complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
Requirement of partial functionality	Not-desired	Optional	Desirable	Critical	Urgent	Nimble
Requirement of change	Seldom	Slow	Moderate	Fast	Rapid	Flashy
Magnitude of change	Insignificant	Minor	Small	Moderate	Large	Extreme
Usability profile	Irregular	Low-stable	Low-high	High-Low	High-stable	High-increase
Usability requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Product size	Very small	Small	Moderate	Large	Very-large	Extreme
Product complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
Interface requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Product mobility requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Expected lifespan	Throwaway	Very-Short	Short	Long	Very-Long	Infinity
Security requirement performance requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Performance requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Funding profile	Irregular	Low-stable	Low-high	High-low	High-stable	High-increase
Funds availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
Staff profile	Irregular	Low-stable	Low-high	High-low	High-stable	High-increase
Staff availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
Access of users	None	Restrictive	Limited	Moderate	Controlled	Free
Technology requirement	Negligible	Scarce	Limited	Adequate	Apple	Abundant
Technology profile	Irregular	Low-stable	Low-high	High-low	High-stable	High-increase
Management capability	Indifferent	Guideline	Flexible	Substantial	Enforced	Exact
Quality assurance and configuration management capability	Trivial	Basic	Intermediate	Substantial	Advanced	Exact

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 therefore, 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. The framework consists of 35 criteria comprising of 6 function point values each as shown in Table 5.

Problem criteria

Maturity of the application: This has to do with the criteria for analyzing the general knowledge of the problem domain to be solved. 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-off).

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

Requirement for partial functionality: This criterion measures, the practicality and/or need to deliver intermediate products that provide only a part of the eventually full functionality of the target product. Values are: C_3 = (not-desirable, optional, desirable, critical, urgent and nimble.

Frequency of change: This criterion estimates the frequency at which the given problem changes. Values are classified into: C_4 = (seldom, slow, moderate, fast, rapid and flashy).

Magnitude of change: This criterion assesses the relative size of expected changes in the problem. Values are: C_5 = (insignificant, minor, small, moderate, large and extreme).

Table 5: Framework for selecting a secured software life cycle models

	Function point values					
Criteria (Ci)	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
Maturity of application domain	Strange	New	Familiar	Standard	Well-Understood	Master Off
Problem complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
Requirement of partial functionality	Not-desirable	Optional	Desirable	Critical	Urgent	Nimble
Frequency of change	Seldom	Slow	Moderate	Fast	Rapid	Flashy
Magnitude of change	Insignificant	Minor	Small	Moderate	Large	Extreme
Security of problem definition	Trivial	Safe	Safer	Secure	Securer	Securest
User-experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced expert
User expression ability	Daft	Indecisive	Silent	Communicative	Expressive	Descriptive
Developer experience in application domain	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced expert
Developers software engineering experience	Novice	Knowledgeable	Experienced	Well-Experienced	Expert	Experienced-expert
System security cognizance	Negligible	Scarce	Limited	Adequate	Apple	Abundant
User involvement and security	Negligible	Scarce	Limited	Adequate	Apple	Abundant
Funding profile	Irregular	Low-Stable	Low-High	High-Low	High-Stable	High-Increase
Funds availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
Staff profile	Irregular	Low-Stable	Low-High	High-Low	High-Stable	High-Increase
Staff availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
Access of users	None	Restrictive	Limited	Moderate	Controlled	Free
Technology profile	Small	Meager	Inadequate	Adequate	Abundant	Copious
Technology availability	Negligible	Scarce	Limited	Adequate	Ample	Abundant
Technology rate	Opportune	Seasonable	Batch	Timely	Well-timed	Online
Independent technology interaction	Negligible	Very-Low	Low	High	Very-High	Highest
Product usability requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Product size	Very Small	Small	Moderate	Large	Very-Large	Extreme
Product complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
Human interface requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Product mobility requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Expected lifespan	Throwaway	Very-Short	Short	Long	Very-Long	Infinity
Security requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Performance requirement	Insignificant	Minor	Useful	Important	Critical	Exacting
Usability profile	Irregular	Low-Stable	Low-High	High-Low	High-stable	High-Increase
Management capability	Indifferent	Guideline	Flexible	Substantial	Enforced	Exact
Quality assurance and configuration management capability	Trivial	Basic	Intermediate	Substantial	Advanced	Exact
Management trust	Negligible	Very-low	Low	High	Very-high	Highest
Management risk	Negligible	Very-low	Low	High	Very-high	Highest
Management security control	Negligible	Very-low	Low	High	Very-high	Highest

We introduce a new criterion, 'Problem Security' that addresses security concerns at the problem definition level. We define it thus:

Security of problem definition: This criterion measures the level of securing the problem definition. 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).

Personnel criteria: These criteria deal with issues relating to the software developers and the users. The criteria and their values are:

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.

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).

Developers experience in application domain: The developers knowledge which could result from being a user in the application domain is evaluated in this criterion assesses using values: C_9 = (novice, knowledgeable, experienced, well experienced, expert and experienced expert).

Developers software engineering experience: This criterion 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 2 new criteria that that addressed security concerns as it affects personnel involved in the Software development process. We define them thus:

System security cognizance: This criterion 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).

User involvement and security: This criterion 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).

Resource criteria: Resource criteria pertain to resources available for development. Resource criteria and their evaluation are:

Funding profile: This criterion 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).

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

Staffing profile: This criterion measures the numbers of people usable over a period of time for a software development project exercise. Values are classified as: C_{15} = (irregular, low-stable, low-high, high-low, high-stable and high-increase).

Staff availability: This criterion estimates the sufficiency of the available staff for a project. Applicable values are: C_{16} = (negligible, scarce, limited, adequate, ample and abundant).

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

Technology profile: This 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 as: C_{18} = (small, meager, inadequate, adequate, abundant and copious).

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).

Technology rate: This criterion measures the rate at which the software development team receives sound and timely information to accomplish their tasks effectively. We introduce the values as: C_{20} = (opportune, seasonable, batch, timely, well-timed and online).

Independent technology interaction: This criterion measures the level of interaction between computer security and operational elements received. In many instances, operational components obtained tend to be far larger and therefore, more influential. We introduce the values: C_{21} = (negligible, very-low, low, high, very-high and highest).

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

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).

Product size: This criterion 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).

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

Human interface requirement: This criterion measures the criticality of the human computer interface. Values are: C_{25} = (insignificant, minor, useful, important, critical and exacting).

Product mobility: This 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).

Expected life span: This criterion 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 infinity).

Product requirement security: This criterion measures the criticality of security in the final product. Values are: C_{28} = (insignificant, minor, useful, important, critical and exacting).

Product performance requirement: This criterion 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.

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).

Organizational criteria: Organizational policy that affects a development effort is analyzed in this study. The organizational criteria are:

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, flexible, substantial, enforced and exact).

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

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).

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).

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 triggers 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 and employing the right software selection criteria for a given project cannot be overemphasized, 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.

ACKNOWLEDGEMENT

We are grateful to Professor E. Onibere and O. Fajuyigbe for their invaluable support and encouragement.

REFERENCES

Akwukwuma, V.N. and A.O. Egwali, 2008. E-Commerce: Online attacks and protective mechanisms. Asian J. Inform. Technol., 7 (9): 394-402.

Alexander, L. and A. Davis, 1991. Criteria for selecting software process models. In: Proceedings of COMPSAC, 09/13/1991-09/13/1991 Japan, pp: 521-528. DOI: 10.1109/CMPSAC.1991.170231.

Boehm, B., 1991. Software risk management: Principles and practices. Software IEEE., 8 (1): 32-41. DOI: 10.1109/52.62930.

Boehm, B. and R. Turner, 2004. Balancing agility and discipline: A guide for the perplexed. Addison-Wesley Professional Edition, Boston. ISBN-10: 0321186125.

Coulouris, G., J. Dollimore and T. Kindberg, 2001. Distributed Systems-Concepts and Design. 3rd Edn. Addison-Wesley Pub. Co., Harlow, England. ISBN-10: 0201619180.

Curtis, B., H. Krasner and N. Iscoe, 1988. A field study of the software design process for large systems. Commun. ACM, 31 (11): 1268-1287.

Davis, A.M., E.H. Bersoff and E.R. Comer, 1988. A strategy for comparing alternative software development life cycle models. Software Eng. IEEE. Trans., 14 (10): 1453-1461. DOI: 10.1109/32.6190.

Little, T., 2005. Context-adaptive agility: Managing complexity and uncertainty. Software IEEE., 22 (3): 28-35. DOI: 10.1109/ms.2005.60.

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

Scacchi, W., 2001. Process models in software engineering, Walt Scacchi, Institute for Software Research, University of California, Irvine. http://www.ics.uci.edu/~wscacchi/Papers/SE-Encyc/Process-Models-SE-Encyc.pdf.

Smith, L.W., 2003. Software Life Cycle, Condensed GSAM Handbook. www.stsc.hill.af.mil/resource/tech-docs/gsam4/chap2.pdf.