Proceedings of National Conference on Emerging Trends: Innovations and Challenges in IT, 19 -20, April 2013

The Unified Process for Network Vulnerability Assessment

Prof. Dhanamma Jagli

Assistant Professor, Department of MCA

V.E.S. Institute Of Technology,

University of Mumbai,India.

dhana1210@yahoo.com

Prof.Shaheen Shaik

Assistant Professor, Department of MCA

Bharati Vidyapeeth's Institute of Management & Information Technology,

University of Mumbai,India.

shaheensk@yahoo.com

Abstract: **Network World - Today's state-of-the-art network security appliances do a great job of keeping the cyber monsters from invading business. As an IT professional, every one needs to know how to perform network security assessments, which relies on the network vulnerability assessment process. In this paper new approach the Unified process for Network vulnerability Assessments hereafter called as a unified NVA is proposed for network vulnerability assessment based on the Unified Software Development Process or Unified Process, it is a popular iterative and incremental software development process framework.**

I. INTRODUCTION

The Unified Modeling Language (UML) is a family of design notations that is rapidly becoming a de facto standard software design language. UML provides a variety of useful capabilities to the software designer, including multiple, interrelated design views, a semiformal semantics expressed as a UML meta model, and an associated language for expressing formal logic constraints on design elements. The primary goal of this work is an assessment of UML's expressive power for modeling software architectures in the manner in which a number of existing software architecture description languages (ADLs) model architectures. This paper presents two strategies for supporting architectural concerns within UML. One strategy involves using UML "as is," while the other incorporates useful features of existing ADLs as UML extensions. As it discuss the applicability, strengths, and weaknesses of the two strategies. The strategies are applied on three ADLs that, as a whole, represent a broad cross-section of present-day ADL capabilities. One conclusion of our work is that UML currently lacks support for capturing and exploiting certain architectural concerns whose importance has been demonstrated through the research and practice of software architectures. In particular, UML lacks direct support for modeling and exploiting architectural styles, explicit software connectors, and local and global architectural constraints.

II. LITERATURE REVIEW

Software architecture is an aspect of software engineering directed at developing large, complex applications in a manner that reduces development costs, increases the potential for commonality among different members of a closely related product family, and facilitates evolution, possibly at system runtime. To date, the software architecture research community has focused predominantly on analytic evaluation of architectural descriptions. Many researchers have come to believe that, to obtain the benefits of an explicit architectural focus, software architecture must be provided with its own body of specification languages and analysis techniques.Mature engineering disciplines are generally characterized by accepted methodical standards for describing all relevant artifacts of their subject matter. Such standards not only enable practitioners to collaborate, but they also contribute to the development of the whole discipline.

1. The Unified Process

The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).Profile of a typical project showing the relative sizes of the four phases of the Unified Process. The Unified Process is not simply a process, but rather an extensible framework which should be

customized for specific organizations or projects. The *Rational Unified Process* is, similarly, a customizable framework. As a result it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

2. *The Unified Process phases:*

The Unified Process has 4 phases as shown in the Fig 1.

1) Inception: Requirements capture and analysis

2) Elaboration: System and class-level design

3) Construction: Implementation and testing

4) Transition: Deployment

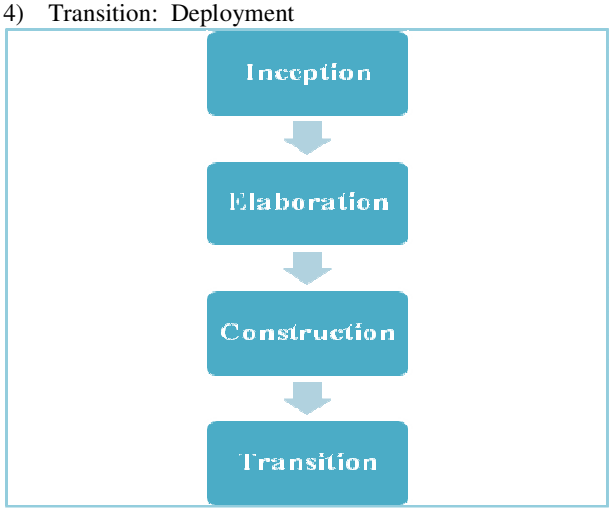

Figure 1: The Unified process Phases

a) Inception Phase

Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.The inception phase, is a preparatory stage that attempts to answer the following questions:

- ✓ What is the purpose of the project?
- ✓ Is the project feasible (e.g. technologically, financially, with current personnel)?
- ✓ Should we buy the project, or build it?
- ✓ Will it be developed now, or built from an existing project?
- ✓ What are the estimated costs?
- ✓ Should we proceed with the project?

The following are typical goals for the Inception phase.

- Establish a justification or business case for the project
- Establish the project scope and boundary conditions
- Outline the use cases and key requirements that will drive the design tradeoffs
- Outline one or more candidate architectures
- Identify risks
- Prepare a preliminary project schedule and cost estimate

The Lifecycle Objective Milestone marks the end of the Inception phase.

Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimate for cost and schedule.This phase mostly deals with project planning and project management This includes Gantt charts and plans, budgets, etc. Since the UP is incremental, the inception phase begins with some activities and produces some initial artifacts, such as a UML diagram or set of diagrams, that is produced during a phase. However, these activities will be continued in later phases and the artifacts modified, and new artifacts added

Some inception artifacts include:

- ❖ *Business case:* Describes high-level goals and constraints, in business terminology
- ❖ *Use-Case model:* Describes functional requirements, and related non-functional req.
- ❖ *Risk management plan:* Describes business, resource, technical, and schedule risks and how to minimize these risks

b) Elaboration Phase

During the Elaboration phase the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address

known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).

The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes core, the most architecturally significant component. It is built in a series of small, timeboxed iterations. By the end of the Elaboration phase the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behaviour in terms of performance, scalability and cost.

The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible; since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.The purpose of the inception phase is to understand the problem. Often 1-3 iterations are required for Elaboration

The main purpose of the elaboration phase is to begin to understand how the software will solve this problem. In other words, this is where much of the architecture and design of the software takes place.

After Elaboration, project risks are essentially eliminated. The user interface has been approved by customers and managers. Technically difficult software components were implemented or proof-of-concept code was created to prove it was possible. Cost estimates are finalized, so budgets can be approved. Preliminary user manuals have been created and analyzed. Analysis, architecture and design well underway after Elaboration

E.g. Use cases should be about 80% complete

The Elaboration phase involves:

- ✓ Continuing work on the use-case model
- ✓ In particular, during the elaboration phase is when activity diagrams might be added to describe how use cases are to operate.
- ✓ diagrams showing how participants (objects and actors) interact during the problem solving process (system sequence diagrams)
- ✓ Prototypes and proof-of-concepts: Some of the more difficult system aspects to accomplish are solved with GUI prototypes, sample database contents, and sample OOPL code for difficult algorithms

c) Construction Phase

Built on the foundation laid in Elaboration. System features are implemented in a series of short, timeboxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration. Common UML (Unified Modelling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction Overview diagrams.

The construction phase involves iterative enhancement to previously created artifacts:

- ✓ Domain model
- ✓ Design model
- ✓ Implementation

Obviously, the implementation is the most significant body of work enhanced during construction

d) Transition Phase

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.

During this phase, the software produced at the end of the construction phase is deployed

This could involve:

- ✓ Rigorous complete system testing
- ✓ Installation programs being purchased or developed
- ✓ Software media being developed

- ✓ Solutions for support/user training being implemented
- ✓ Best testing under varied deployment environments
- ✓ Verifying that the system meets acceptance criteria

3. The Unified Process Is Iterative and Incremental

Developing a commercial software product is a large undertaking that may continue over several months to possibly a year or more. It is n iteration that n iteration that practical to divide the work into smaller slices or mini-projects .Each mini-project is an iteration that results in an increment. Iteration refers to steps in the workflow, and increments to growth in product. To be most effective, the iterations must be controlled; that is they must be selected and carried out in a planned way. This is why they are mini-projects.

Developers base the selection of what is to be implemented in iteration upon two factors. First, the iteration deals with a group of uses cases that together extend the usability of the product as developed so far. Second, the iteration deals with the most important risks. Successive iterations build on the development artifacts from the state at which they were left at the end of the previous iteration. It is a mini-project, so from the use cases it continues through the consequent development work-analysis, design, implementation, and test-that realizes in the form of executable code the use cases being developed in the iteration. Of course, an increment is not necessarily additive. Especially in the early phases of the life cycle, developers may be replacing a superficial design with a more detailed or sophisticated one. In later phases increments are typically additive.

In every iteration, the developers identify and specify the relevant use cases, create a design using the chosen architecture as a guide, implement the design in components, and verify that the components satisfy the use cases. If iteration meets its goals—and it usually does—development proceeds with the next iteration. When iteration does not meet its goals, the developers must revisit their previous decisions and try a new approach.

To achieve the greatest economy in development, a project team will try to select only the iterations required to reach the project goal. It will try to sequence the iterations in a logical order. A successful project will proceed along a straight course with only small deviations from the course the developers initially planned. Of course, to the extent that unforeseen problems add iterations or alter the sequence of iterations, the development process will take more effort and time. Minimizing unforeseen problems is one of the goals of risk reduction.

There are many benefits to a controlled iterative process:

- Controlled iteration reduces the cost risk to the expenditures on a single increment. If the developers need to repeat the iteration, the organization loses only the misdirected effort of one iteration, not the value of the entire product.
- Controlled iteration reduces the risk of not getting the product to market on the planned schedule. By identifying risks early in development, the time spent resolving them occurs early in the schedule when people are less rushed than they are late in the schedule. In the "traditional" approach, where difficult problems are first revealed by system test, the time required to resolve them usually exceeds the time remaining in the schedule and nearly always forces a delay of delivery.
- Controlled iteration speeds up the tempo of the whole development effort because developers work more efficiently toward results in clear, short focus rather than in a long, ever-sliding schedule.
- Controlled iteration acknowledges a reality often ignored—that user needs and the corresponding requirements cannot be fully defined up front. They are typically refined in successive iterations. This mode of operation makes it easier to adapt to changing requirements.

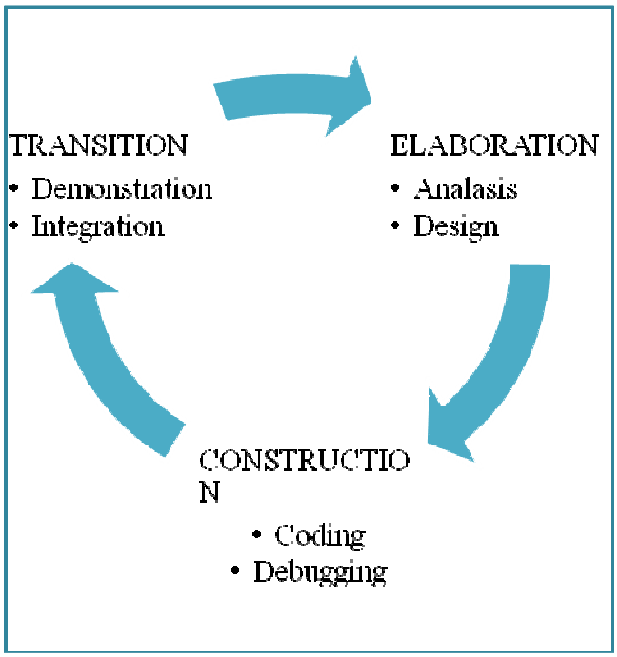

Figure 2: The Iterative Unified process

The successful commercialization of many applications of wireless networks relies on the assurance of the *confidentiality* and *integrity* of the data communicated through the network. Confidentiality is defined as the ability to keep data secret from all but a set of authorized entities, and integrity is defined as the ability to verify that data has not been maliciously or accidentally altered while in transit Recent research has demonstrated that these properties can be efficiently compromised by physically capturing network nodes and extracting cryptographic keys from their memory Such *node capture attacks* are possible due to the potential unattended operation of wireless nodes and the prohibitive cost of tamper-proof hardware in portable devices. Using the cryptographic keys recovered in a node capture attack, an adversary can compromise the confidentiality and integrity of any messages secured using the compromised keys.

4. Vulnerability

Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw.[1] To exploit vulnerability, an attacker must have at least one applicable tool or technique that can connect to a system weakness. In this frame, vulnerability is also known as the attack surface.

a) Vulnerability Types

Vulnerabilities are classified according to the asset class they are related to:

1. Hardware
    a. susceptibility to humidity
    b. susceptibility to dust
    c. susceptibility to soiling
    d. susceptibility to unprotected storage
2. Software
    a. insufficient testing
    b. lack of audit trail
3. Network
    a. unprotected communication lines
    b. insecure network architecture
4. Personnel
    a. inadequate recruiting process
    b. inadequate security awareness
5. Site
    a. area subject to flood
    b. unreliable power source
6. Organizational
    a. lack of regular audits
    b. lack of continuity plans
    c. lack of security

b) Identifying and removing vulnerabilities

Many software tools exist that can aid in the discovery (and sometimes removal) of vulnerabilities in a computer system. Though these tools can provide an auditor with a good overview of possible vulnerabilities present, they can not replace human judgment. Relying solely on scanners will yield false positives and a limited-scope view of the problems present in the system.

Vulnerabilities have been found in every major operating system including Windows, Mac OS, various forms of UNIX and Linux, OpenVMS, and others. The only way to reduce the chance of a vulnerability being used against a system is through constant vigilance, including careful system maintenance (e.g. applying software patches), best practices in deployment (e.g. the use of firewalls and access controls) and auditing (both during development and throughout the deployment lifecycle).

III. THE PROPOSED MODEL

The Unified NVA is the Unified process for Network vulnerability assessment is an efficient approach for network vulnerabilities findings that can be exploited by a determined intruder to gain access to or shut down a network. Network vulnerability is a condition, a weakness of or an absence of security procedure, or technical, physical, or other controls that could be exploited by a threat

1. The Unified NVA Phases

The Unified NVA has 4 phases as shown in the Fig 3.

1) Initiation

2) Expansion

3) Erection

4) Evolution

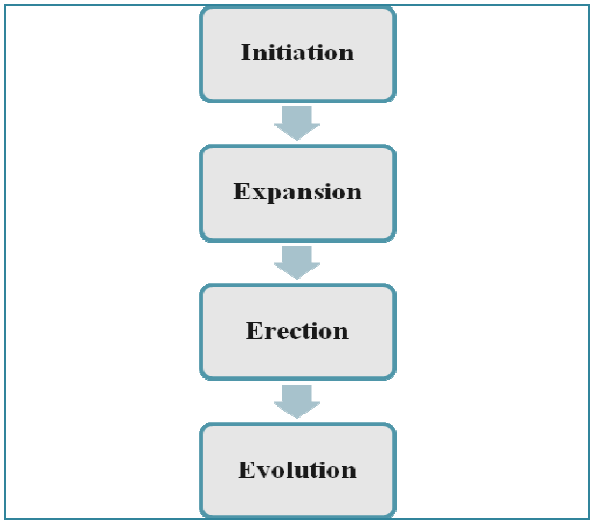

Figure 3: The Unified NVA Phases

a) Initiation Phase

Risk analysis is the initiation for NVA. Assessing risk is a process and as such, is something that must be periodically repeated. It's really not much different from the automated patch-management tools are probably using. True security requires ongoing effort. There is never a wrong time to assess risk and examine network vulnerabilities.

There are four ways in which can respond to risk: *avoidance*, *transference*, *mitigation*, and *acceptance*:

b) Expansion Phase

Policy assessments— will call this a level I assessment. It is a top-down look at the organization's policies, procedures, and guidelines. This type of vulnerability assessment does not include any hands-on testing. The purpose of a top-down policy assessment is to answer three questions:

- ✓ *_ Do the applicable policies exist?*
- ✓ *_ Are they being followed?*
- ✓ *_ Is there content sufficient to guard against potential risk?*

c) Erection Phase

Rolling out new policy may seem to be no more difficult than dictating the date of expected compliance, but that is not the case. Consider a major change to the authentication and authorization policy. Mandate that everyone must change to complex passwords on the first day of the next month. So, on Monday morning, all r employees attempt to change passwords and many experience problems. The result is that the help desk is flooded with calls and many individuals experience an unproductive morning, waiting to log in and begin work. Policies should be implemented in such a way that the change is gradual, staged, or piloted. Many individuals already have the belief that security policies inhibit work and slow things down, so want to make sure that any change make does not contribute to that sentiment.

d) Evolution Phase

Penetration tests— unlike assessments and evaluations, penetration tests are adversarial in nature. Will refer to

penetration tests as level III assessments. These events typically take on an adversarial role and look to see what the outsider can access and control. Penetration tests are much less concerned with policies and procedures and are more focused on finding "low-hanging fruit" and seeing what a hacker can compromise on this network. We almost always recommend that organizations complete an assessment and evaluation before beginning a penetration test, because a company with adequate policies and procedures can't implement real security without documented controls. The general NVA life cycle as shown in the below Fig 4.

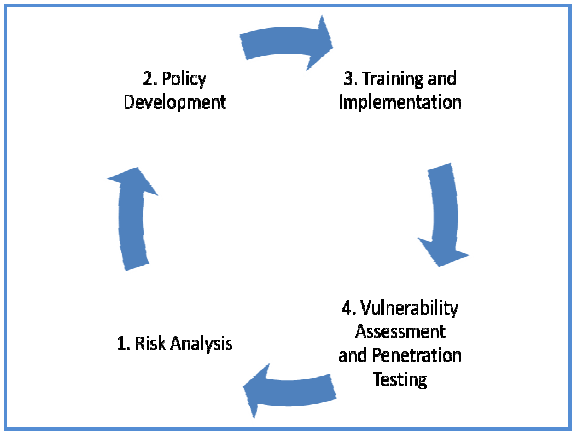

Figure 4: The NVA Life Cycle

IV. SYSTEM ARCHITECTURE

Vulnerability management is the cyclical practice of identifying, classifying, remediating, and mitigating vulnerabilities this practice generally refers to software vulnerabilities in computing systems. Successive iterations build on the development additive from the state at which they were left at the end of the previous iteration as shown in the below Fig 5.

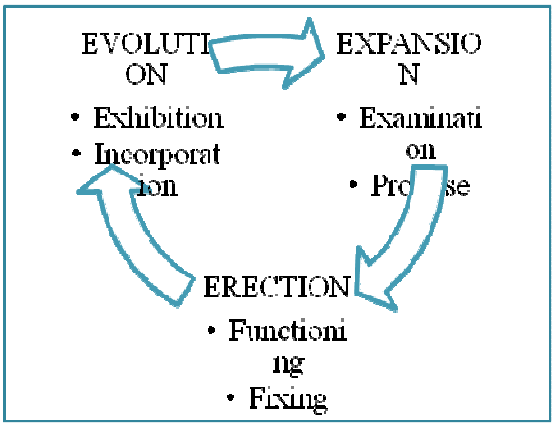

Figure 5: The Iterative Unified NVA

V.

Network Vulnerability Assessor base the selection of what is to be done in iteration upon two factors. First, the iteration deals with a group of policies that together extend the usability of the network resources.. Second, the iteration deals with the most important awareness or training to users.

VI. CONCLUSION

The unified Network vulnerability Asseemnet model is implemented successfully. This model as an iterative process will help to increase efficiency and reduce the vulnerabities in the network resources and protect network assets from attackers.

References

[1]. "Modeling Software Architectures in the Unified Modeling Language", by NENAD MEDVIDOVIC University of Southern California DAVID S. ROSENBLUM and DAVID F. REDMILES University of California, Irvine and JASON E. ROBBINS CollabNet, Inc.

[2]. The text book, "Inside Network Security Assessment: Guarding your IT Infrastructure", by By Michael Gregg, David Kim.

[3]. The text book,"The Unified Software Development Process ", By Ivar Jacobson, Grady Booch, James Rumbaugh, Pearson Education.

[4]. K. Jain, "Security based on network topology against the wiretapping attack," IEEE Wireless Communication, vol. 11, no. 1, pp. 68–71, Feb.2004.

[5]. D. Liu, P. Ning, and R. Li, "Establishing pairwise keys in distributed sensor networks," ACM Trans. Information and System Security, vol. 8,no. 1, pp. 41–77, Feb. 2005.

[6]. W. A. Arbaugh, W. L. Fithen, and J. McHugh. Windows of Vulnerability: a Case Study Analysis. IEEE Computer, 2000.

[7]. Agarwal, R., & Sinha, A. P. (2003). Object-Oriented Modeling with UML: A Study of Developers' Perceptions. Communications of the ACM, 46(9), 248-256.

[8]. Atkinson, C., & Kühne, T. (2002). Rearchitecting the UML Infrastructure. ACM Transactions on Modeling and Computer Simulation, 12(4), 290-321.

[9]. Barbier, F., Henderson-Sellers, B., Parc-Lacayrelle, A. L., & Bruel, J.-M. (2003). Formalization of the Whole-Part Relationship in the Unified Modeling Language. IEEE Transactions on Software Engineering, 29(5), 459-470.

[10] www.Wikipedia.com