AN ANALYZER-BASED SECURITY MEASUREMENT MODEL FOR INCREASING SOFTWARE SECURITY

Sen-Tarng Lai

Department of Information Technology and Management,
Shih Chien University, Taipei, Taiwan
stlai@mail.usc.edu.tw

ABSTRACT

Software security has become an increasingly important issue for information and software system. Secure vulnerabilities of software system may cause a company out of business and even destroy the social normal operation. How to improve software security becomes a critical issue in software development process. In this paper, utilizing the static program analyzer and dynamic simulation analyzer to collect metrics, proposes an Analyzer-based Software Security Measurement (ASSM) model. Applying ASSM model, the secure flaws of software system can be identified clearly. And, using a Rule-based Software Security Improvement (RSSI) operation to control and improve security defects and security vulnerability of software system. The security risk of software system can be reduced efficiently.

KEYWORDS

Software Security, Security vulnerability, Security Metric, Measurement Model, ASSM

1. INTRODUCTION

Influence of software security requirements overtakes the functional and performance requirements for the software system [1, 2]. It is because of security vulnerability of software system may cause a company out of business and even destroy the social normal operation. There are many factors may cause software security vulnerability. However, according to the formed reasons of security vulnerability, two kinds of security flaws of software system are specified as (1) the negligence of program implementation causes the security vulnerability; [3] (2) the plan defects of operation environment causes the security vulnerability [1]. Any kinds of prevention of security flaws have to invest much manpower and cost. Implementing high secure program need high development cost, that make developer and user without respect to invest more development cost. How to get the break-even point between secure software investment and development cost becomes a valuable exploration issue [3, 4].

Security vulnerability of software system cause the enterprise loss and crisis which is hard to expect and evaluate. In order to avoid the serious damage of software security vulnerability, it is necessary to concern the problems of security flaws in software development process [5, 6]. Three major technologies for identifying and correcting the security flaws of software system: (1) code inspection [7, 8]; (2) testing [9]; (3) static and dynamic analyzer [10, 11, 12]. Code inspections are expensive and time consuming approach but can not find security flaws automatically. Software testing phase is important step to find the defects and problems of software system. However, software test phase concentrates on the functionality, integration and execution performance but omits the security flaws of software system. Static and dynamic analyzer can automatically identify the security flaws which are defined by the user. In software implementation, static analyzer can help us identify and collect the security flaws of implementation phase. In software operation, dynamic analyzer can help us identify and collect the security flaws of maintenance and operation phase.

In software development process, many security flaws may cause software security vulnerability [5]. Most of security vulnerability of software system are depended on the defects of program implementation and the problems of operation environment. Security defects and problems of program implementation always are omitted that makes the longest and most serious influence for the software system. Therefore, collecting over all metrics of security flaws and parsing all kinds of security vulnerability, the revision and improvement operations can be proposed in time. The security vulnerability and risks of software system can be reduced. Based on static program analyzer and dynamic simulation analyzer, an Analyzer-based Software Security Measurement (*ASSM*) model is proposed in this paper. Applying ASSM model, software security flaws can be identified, and the crisis of software security can be controlled and improved. The secure risk of software system can be reduced and security of software system can be increased efficiently. In Section 2, discusses the security flaws and defects which cause from program implementation and operation environment. Quantitative secure factors are the critical tasks for identifying the security flaws and problems. In Section 3, describes the metrics collection and normalization of software security. In Section 4, based on the normalized security metrics, proposes an *ASSM* model. The model can identify the problems and defects of software security in software development process. In Section 5, applying the formulas of ASSM model, proposes a Rule-based Software Security Improvement (RSSI) operation. Finally, summary and future works are given in the last Section.

2. SECURITY OF SOFTWARE SYSTEM

There are many factors which may cause software security vulnerability. Program implementation and operation environment are two critical factors which may cause high impact of software security defects.

2.1 Security program implementation

Program specification is a major product of detailed design for implementation phase. According to the program specification, programmer transfers it into the source code in implementation phase. However, the program specification designer concentrates on the functionality, input/output interface and operation logic, but almost neglects security of program implementation. Security flaws of program implementation are not discussed in program specification. In program implementation, security flaws are embedded into the software products and become the security vulnerability to affect the security of software system [2, 13]. Secure software must concentrate on the security flaws which may be caused security vulnerability in program implementation. There are several papers discuss static analyzer for programming security [10, 11, 12]. Using static analyzer can reduce dynamic testing cost and identify security defects early. Six critical secure items can robust enhance security of software system in program implementation (shown in Figure 1):

(1) Secure arithmetic expressions: Many security flaws may happen in arithmetic expression. The security flaws of arithmetic expression include divide by zero, overflow of arithmetic operation, misused data type of arithmetic expression, and truncation error of arithmetic operation. For avoiding the security flaws of arithmetic expression, the arithmetic expression should be implemented complete in coding phase.

(2) Secure control operations: Many security flaws may happen in arithmetic expression. The security flaws of control operations include infinite loop, deadlock situations and boundary

defects. For avoiding the security flaws of control operations, the control condition should be considered complete in coding phase.

(3) Secure array index: Many security flaws may happen in array index. The security flaws of array index include index out of range, incorrect index variable and uncontrollable array index. For avoiding the security flaws of array index, the array index should be checked complete in coding phase

(4) Secure data format: Many security flaws may happen in data format. The security flaws of data format include mismatched data contents, mismatched data type, mismatch data volume. For avoiding the security flaws of data format, the input file format should be checked complete in coding phase.

(5) Secure resource allocation: Many security flaws may happen in resource allocation. The security flaws of resource allocation include exhausted memory, unreleased memory exhausted resources, and unreleased resources. For avoiding the security flaws of resource allocation, the resource allocation should be checked complete in coding phase.

(6) Secure component interface: Many security flaws may happen in component interface. The security flaws of component interface include inconsistent parameter numbers, inconsistent data types, and inconsistent size of data types. For avoiding the security flaws of component interface, the component interface parameters should be checked complete in coding phase.

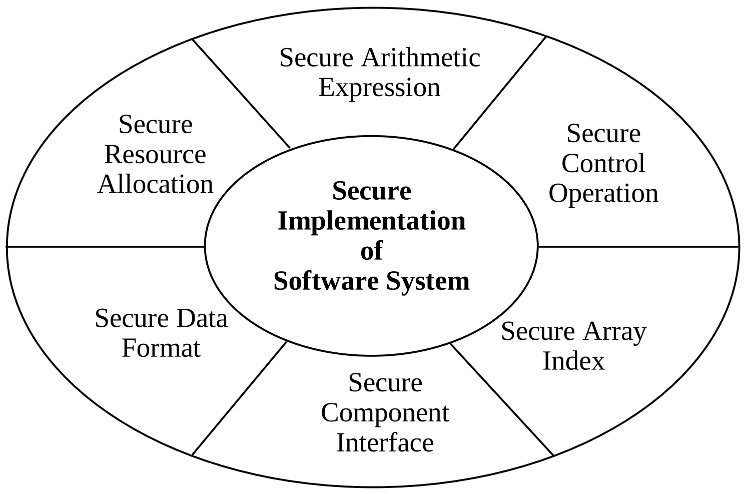
Secure Arithmetic Expression
Secure Resource Allocation
Secure Control Operation
Secure Implementation of Software System
Secure Data Format
Secure Array Index
Secure Component Interface

Figure 1. A frame of secure implementation of software system

2.2 Security operation environment

According to software development life cycle (SDLC), the released software system will enter into the maintenance and operation phase. The software system with security flaws may cause high security risks and unexpected result in software operation. Four critical secure items can robust enhance security of software system in operation environment (shown in Figure 2):

(1) Secure user authority: A released software system represents that the system is in the executable state. Many users can use the released system to do specific jobs. However, user privilege definition, user password management and user detailed information maintenance are important tasks to control and manage the user authority. Without user authority control and management, the operation environment of software system will be on high security risk.

(2) Secure kernel operation: A released software system must be adapted to a suitable operating system and environment. The kernel of operating system and environment become a critical

factor to affect the operating security of software system. Resource management, execution file protection, main memory control and process scheduling are major tasks of operating system. The security flaws, which embedded in the tasks of operating system, become high security risk for software system operation.

(3) Secure network planning: Network environment is a critical tool in the E-commerce years. It is necessary for a released software system to adapt to the network environment. However, data transmission management, remote access control, and transmission contents always become the security vulnerability of software system operation.

(4) Secure I/O management: File is an important media which can store critical information and record log data. However, file privilege definition, file access management, file contents, file organization and file size are major factors to affect the security software operation. Creating robust I/O management procedure can reduce the security vulnerability in software operation.

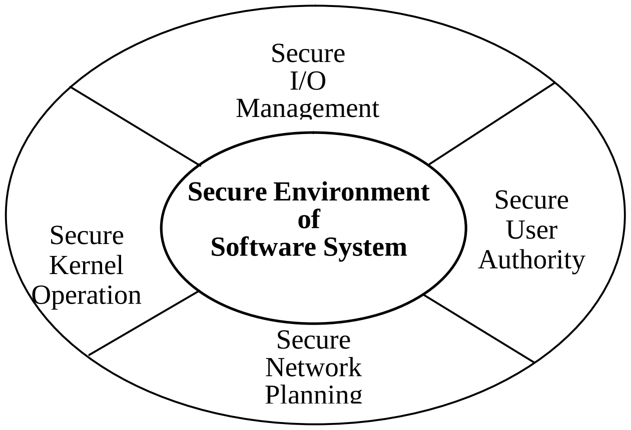

Figure 2. A frame of secure environment of software system

3. SECURITY METRICS COLLECTION AND NORMALIZATION

Security flaws were embedded into the software system in software development. Collecting and analyzing the security metrics can help us to identify the security flaws.

3.1 Static-based security metrics

In software development, functional requirements are the important and primary missions which have to be finished fully [9]. However, the end user is not care about the software security, and induces that the secure requirements have not be defined and specified in requirement specification. The requirement specification without secure requirements may cause that many security flaws are embedded into the software system in implementation phase [14]. After software system is released, the security flaws become unexpected security vulnerability which affect system operation. In order to control and reduce secure risk, the security of software system should be measured. Static program analyzer can help us collect major metrics of software security flaw [3, 12]. Based on six critical security flaws of implementation phase, the metric collection approach are described as follows:

(1) Three secure check factors for collecting the security metric of arithmetic expression:

- Utilize security check to prevent a divisor from being a zero in division operation of arithmetic expression?
- Utilize security check to prevent the variable from overflowing in the arithmetic operation?
- Utilize security check to prevent the arithmetic truncation from causing the system abnormal behavior and security flaw?

(2) Four secure check factors for collecting the security metric of control operation:
- Utilize security check to prevent the iteration statements from causing infinite loop in program execution?
- Utilize security check to prevent the waiting statements from causing deadlock situations in program execution?
- Utilize security check to confirm the input files are on the readable state?
- Utilize security check to confirm the output files are on the writable state?

(3) Two secure check factors for collecting the security metric of array index:
- Utilize security check to prevent the range of array index from outing of range in program execution?
- Utilize security check to confirm the data type of index variable of array is an integer data type?

(4) Two secure check factors for collecting the security metric of data format:
- Utilize security check to confirm the data contents to match with the data type of input format in coding phase?
- Utilize security check to confirm input data type to meet the interface definition of program?

(5) Two secure check factors for collecting the security metric of resource allocation:
- Utilize security check to confirm memory allocation statement to concern exhausted memory state?
- Utilize security check to prevent the resource allocation statement from referred to the locked resource state?

(6) Three secure check factors for collecting the security metric of component interface:
- Utilize security check to prevent number of parameters between two components from inconsistent error?
- Utilize security check to prevent data types between two components from inconsistent error?
- Utilize security check to prevent size of data types between two components from inconsistent error?

3.2 Dynamic-based security metrics

A released software system belongs to the open operation environment. End user, which passed the authority checking, can use the software system in specific period. For keeping the released software system working smoothly and normally, the security vulnerability should be identified and controlled efficiently. It may reduce many unexpected situations, which cause by secure leaks problems in software operation. Dynamic-based security metric collection is applied to the execution state of software system. Simulation approach can help us collect four major metrics in system operation. Dynamic-based security flaws metrics collection for four simulation approaches is described as follows:

(1) Three secure f check actors for collecting the security metric of user authority:
- Utilize security strategy to prevent users from using illegal privilege?
- Utilize security strategy to prevent different level users from violating access rules in software operation?
- Utilize security strategy to prevent the software system from losing user authority control and management?

(2) Three secure check factors for collecting the security metric of kernel control:
- Utilize security strategy to prevent the kernel from losing memory management and control?
- Utilize security strategy to prevent the kernel from losing CPU management and control?
- Utilize security strategy to prevent the kernel from losing peripheral devices management and control?

(3) Three secure check factors for collecting the security metric of network planning:

- Utilize security strategy to prevent the network from crashing?
- Utilize security strategy to prevent that the virus is transmission from the network?
- Utilize security strategy to prevent the incorrect, incomplete or inconsistent information from transmitting by the network?

(4) Three secure check factors for collecting the security metric of I/O management:

- Utilize security strategy to prevent file storage from destroying by the unauthorized users?
- Utilize security strategy to prevent backup mechanism from destroying by the unauthorized users?
- Utilize security strategy to prevent file recovery mechanism from destroying by the unauthorized users?

3.3 Security metrics normalization

Generally, a potential quality characteristic is combined with several primitive metrics [15, 16, 17]. Some primitive metrics, which are concerned with the security factors of software system, have different scale measurement values in their representation. To combine these primitive metrics, which have different scale values in their representation, we recommend that all measure scale values of each primitive metric should be normalized to a value between 0 and 1. Close to 1 represents the most desirable value, and close to 0 represents the least desirable value [18]. After normalization, the properties of correctness, completeness, and consistency for different primitive metrics should still be kept.

4. SOFTWARE SECURITY MEASUREMENT MODEL

Primitive metrics play an individual role to measure the individual characteristic of software security. For measuring the overall software security, the primitive metrics of software security should be normalized and combined.

4.1 The linear metric combination model

Two kind of metric combination models are Linear Combination Model (LCM) and NonLinear Combination Model (NLCM). NLCM has high accuracy measurement than LCM. However, LCM has high flexibility, more extensible and easy formulation than NLCM (shown in Table 1). For this, LCM is applied to security metrics combination in this paper. There are three linear combination models for metric combination [18]:

Table 1. Features comparison of two Combination Models

Models / **Features**	Linear Combination Model	NonLinear Combination Model
Accuracy	*Middle*	*High*
Flexibility	*High*	*Low*
Extensibility	*High*	*Low*
Formulation	*Easy*	*Difficult*

(1) Equally Weighted Linear Combination: This is the simplest combination model. Each primitive metric has an equal weight constant to combine into the higher-level metric.

(2) Unequally Weighted Linear Combination: In this model, according to the optimistic and

pessimistic predications, different weights are assigned to different primitive metrics to combine into the higher-level metric.

(3) Dynamically Weighted Linear Combination: In this model, we can assign different weight values to the same primitive metric in different situations to combine into the higher-level metric (HLM). It means that the model has more flexibility and practicability than the other two models.

In this paper, Dynamically Weighted Linear Combination is selected for metric combination.

4.2 Analyzed-based software security measurement model

In Section 3, several primitive metrics were identified to measure security of software system. The primitive metrics must be combined for the high level security measurements of software system. For example, security metrics of arithmetic expression is combined with three low level secure factors and security metrics of control operation is combined with four low level secure factors. For this, a metrics combination model that is based on the dynamically weighted linear combination [5, 7] is proposed. In the model, six primitive metrics are combined into program implementation security measurement (PISM). And four primitive metrics are combined into operation environment security measurement (OESM) as follows:

(1) Combine arithmetic expressions, control operations, array index, input format, resource allocation and component interface metrics into *PISM* shown as Formula (1).

PISM: Program Implementation Security Measurement

AESM: Security Metrics of Arithmetic Expression	*W_{aesm}: Weight of AESM*
COSM: Security Metrics of Control Operation	*W_{cosm}: Weight of COSM*
AISM: Security Metrics of Array Index	*W_{aism}: Weight of AISM*
IFSM: Security Metrics of Input Format	*W_{ifsm}: Weight of IFSM*
RASM: Security Metrics of Resource Allocation	*W_{rasm}: Weight of RASM*
CISM: Security Metrics of Component Interface	*W_{cism}: Weight of CISM*

$$PISM = W_{aesm}* AESM + W_{cosm}* COSM + W_{aism}*AISM + W_{ifsm}*IFSM+ W_{rasm}*RASM + W_{cism}*CISM$$
$$W_{aesm} + W_{cosm} +W_{aism}+ W_{ifsm}+ W_{rasm} + W_{cism} =1 \qquad (1)$$

(2) Combine user authority, kernel operation, network environment and I/O management security measurement into OESM shown as Formula (2).

OESM: Operation Environment Security Measurement

UASM: Security Metrics of User Authority	*W_{uasm}: Weight of UASM*
KOSM: Security Metrics of Kernel Operation	*W_{kosm}: Weight of KOSM*
NESM: Security Metrics of Network Environment	*W_{nesm}: Weight of NESM*
IOMSM: Security Metrics of I/O Management	*W_{iomsm}: Weight of IOMSM*

$$SOSM = W_{uasm}*UASM + W_{kosm}*KOSM + W_{nesm}*NESM+ W_{iomsm}*IOMSM$$
$$W_{uasm} + W_{kosm} +W_{nesm}+ W_{iomsm} =1 \qquad (2)$$

Finally, two security measurements *(PISM, OESM)* are combined into a security indictor of software system (*SISM*) shown as Formula (3).

SISM: Security Indictor of Software System

PISM: Program Implementation Security Measurement *W_{pism}: Weight of PISM*

OESM: Operation Environment Security Measurement W_{oesm}: *Weight of OESM*

$$SISM= W_{pism}*PISM + W_{oesm}*OESM \qquad W_{pism}+ W_{oesm} = 1 \qquad (3)$$

The measurement scheme for software security is called an Analyzer-based Software Security Measurement (*ASSM*) Model (see Figure 3). Based on the criteria of software security, the formulas of ASSM model can help us to identify low security measurements and derive from low security metrics. In software development process, low security metrics represent that several security flaws are incorporated to the software system. Applying ASSM model, software security flaws can be identified, and the crisis of software holes can be controlled and improved.

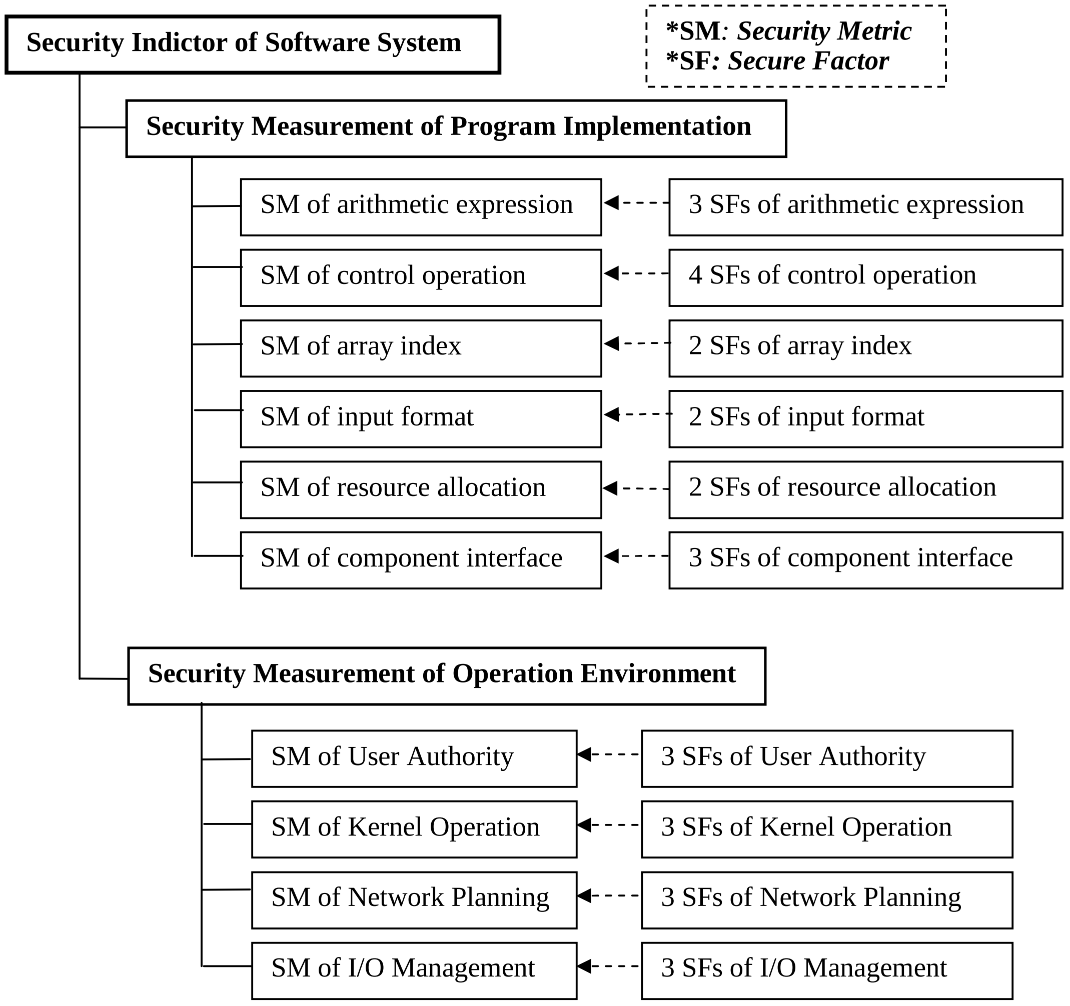
Security Indictor of Software System
*SM: Security Metric
*SF: Secure Factor
Security Measurement of Program Implementation
SM of arithmetic expression
3 SFs of arithmetic expression
SM of control operation
4 SFs of control operation
SM of array index
2 SFs of array index
SM of input format
2 SFs of input format
SM of resource allocation
2 SFs of resource allocation
SM of component interface
3 SFs of component interface
Security Measurement of Operation Environment
SM of User Authority
3 SFs of User Authority
SM of Kernel Operation
3 SFs of Kernel Operation
SM of Network Planning
3 SFs of Network Planning
SM of I/O Management
3 SFs of I/O Management

Figure 3. A frame of the Analyzer-based Software Security Measurement Model

5. Rule-based Approach to improve Software Security

Software security indicator is measured by the ASSM model. ASSM model is combined from three layers quantitative data which are secure factors, primitive metrics and high level measurements. Based on software security indicator, the security problems and defects can be identified. Therefore, if quantitative indicator is under the criteria, then Formula (3) should be checked to identify the defects of program implementation and operation environment. If security measurement of program implementation is under the criteria, then Formula (1) should be checked to identify the defects of six critical security items of program implementation. If security measurement of operation environment is under the criteria, then Formula (2) should be checked to identify the defects of four critical security items of operation environment. Applying the formula of ASSM model, a Rule-based Software Security Improvement (RSSI) operation is proposed. Five production rules of RSSI operation are described as follows:

Rule 1:

IF software security indicator is under Software Security Criteria
THEN the measurement of program implementation and the measurement of operation environment should be checked. Based on the Formula (3), two high level measurements can be analyzed to identify the software security problems and defects.

Rule 2:

IF measurement of program implementation is under Program Security Criteria
THEN the security metrics of arithmetic expression, control operations, array index, data format, resource allocation and component interface should be checked. Based on the Formula (2), six primitive level metrics can be analyzed to identify security problems and defects of program implementation.

Rule 3:

IF measurement of operation environment is under Environment Security Criteria
THEN the security metrics of user authority, kernel operation, network planning and I/O management should be checked. Based on the Formula (3), for primitive level measurements can be analyzed to identify the security problems and defects of operation environment.

Rule 4:

IF security metric of program implementation is under Metric Security Criteria
THEN the low level secure factors contrast to security metrics of program implementation should be checked. Based on the Combination Formula, each low level factor can be analyzed to identify the problems and defects of implementation secure factor.

Rule 5:

IF security metric of operation environment is under Metric Security Criteria
THEN the low level secure factors contrast to the security metrics of operation environment should be checked. Based on the combination Formula, each secure factor of operation environment can be analyzed to identify the problems and defects of environment.

Based on ASSM model, RSSI operation can identify the security problems and defects of implementation phase and testing phase (see Figure 4). And, the security defects and security vulnerability of software system can be efficiently controlled and improved. The security vulnerability of software system can be reduced and security of software system can be concretely enhanced.

6. Conclusions

Static program analyzer can collect secure factors of security flaws in program implementation. Dynamic simulation analyzer can collect secure factors of security defects in software operation. In this paper, several primitive metrics for measuring the security of software system are surveyed and discussed. For measuring the overall software security, the primitive metrics of software security should be normalized and combined. In this paper, an Analyzer-based Software Security

Measurement (ASSM) model is proposed. ASSM model has the features of high flexibility, high extensibility and easy formulation. Applying ASSM model, RSSI operation can help identify the security defects of implementation phase and testing phase. The crisis of software holes can be controlled and improved. The software system security can be enhanced. Finally, our future work is to collect and analyze the feedback data to improve the ASSM model.

ACKNOWLEDGEMENTS

This research was supported by the NSC funds (Project No.: NSC 101-2221-E-158 -007) and Shih Chien University 2012 research project funds (Project No.: 101-05-04010) .

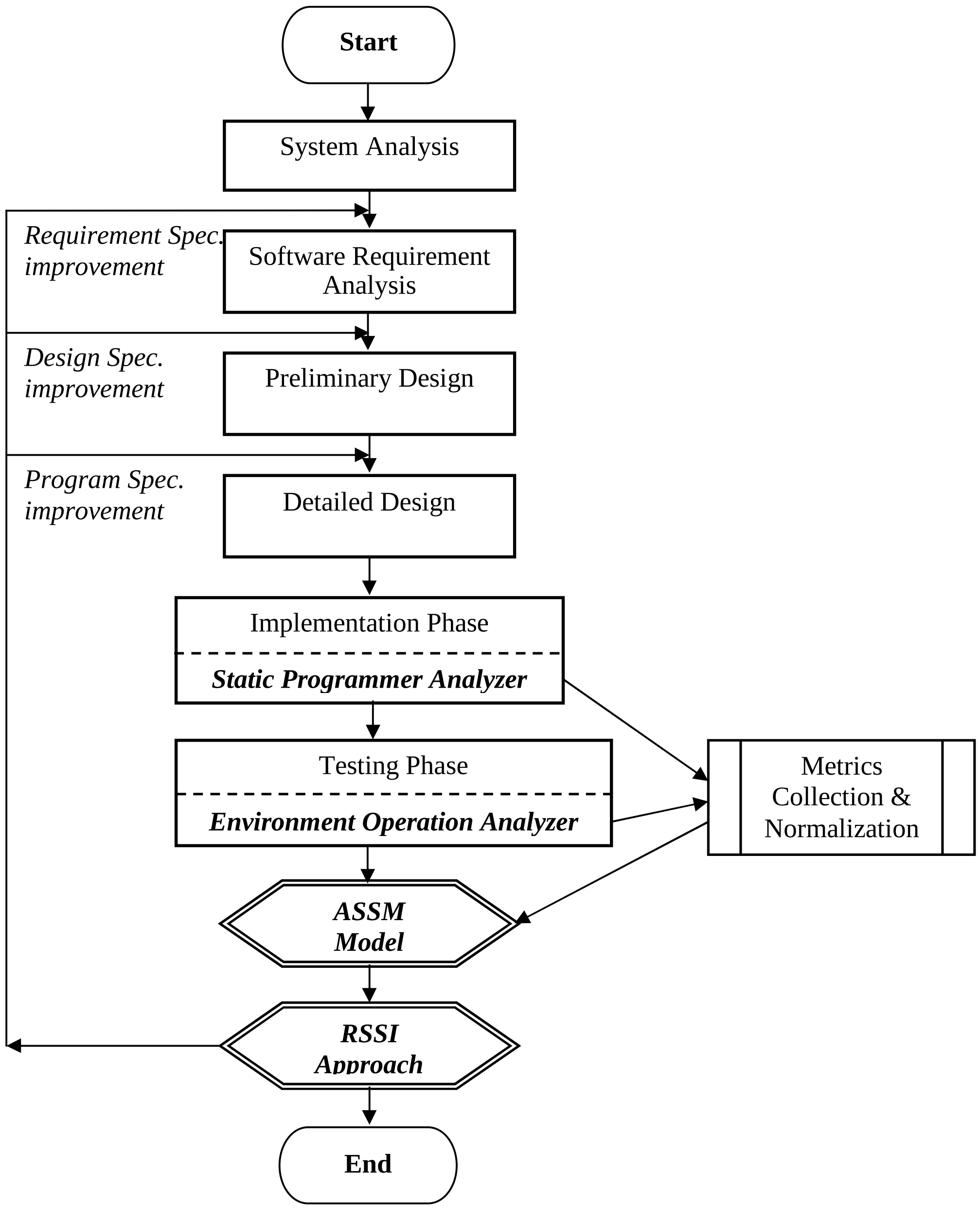
Start
System Analysis
Requirement Spec. improvement
Software Requirement Analysis
Design Spec. improvement
Preliminary Design
Program Spec. improvement
Detailed Design
Implementation Phase
Static Programmer Analyzer
Testing Phase
Environment Operation Analyzer
Metrics Collection & Normalization
ASSM Model
RSSI Approach
End

Figure 4. A flowchart of the Software Security Improvement Operation

REFERENCES

[1] McGraw, G. (2006) Software Security – Building Security In, Addison-Wesley.

[2] Hall, A., and Chapman, R. (2002) "Correctness by Construction: Developing a Commercial Secure System," IEEE Software, January/February, pp.18-25.

[3] Evans, D., and Larochelle, D. (2002) "Improving Security Using Extensible Lightweight Static Analysis," IEEE Software, January/February, pp.42-51.

[4] Cowan, C. (2003) "Software Security for Open-Source Systems," IEEE Security & Privacy, Vol. 1, No. 1, pp. 38–45.

[5] Apvrille, A. and Pourzandi, M. (2005) "Secure Software Development by Example," IEEE Security & Privacy, Vol. 3, No. 4, pp. 10-17.

[6] Viega, J. and McGraw, G. (2002), Building Secure Software, Addison-Wesley.

[7] Pressman, R. S. (2010), Software Engineering: A Practitioner's Approach, McGraw-Hill, New York.

[8] Fairley, R. (1985) Software Engineering Concepts, McGraw-Hill, Inc.

[9] Myers, G. J. (1979) The Art of Software Testing, John Wiley & Sons, Inc., New York.

[10] Walden, J. and Doyle, M. (2012) "SAVI: Static-Analysis Vulnerability Indicator," IEEE Security & Privacy, Vol. 10, No. 3, pp. 32-39.

[11] Chess, B. and West , J. (2007) Secure Programming with Static Analysis, Addison-Wesley Professional.

[12] Bush, W. R., Pincus, J. D.and Siela, D. J. (2000) "A Static Analyzer for Finding Dynamic Programming Errors," Software Practice and Experience, Vol. 30, June, pp. 775-802.

[13] Howard, M. and Le Blanc, D. (2002) "Writing Secure Code," Microsoft Press.

[14] Bishop, M. (2003) Computer Security: Art and Science, Addison-Wesley.

[15] Boehm,B., W. (1981) Software Engineering Economics, Prentice-Hall, New Jersey.

[16] Conte,S. D., Dunsmore, H. E. and Shen, V. Y. (1986) Software Engineering Metrics and Models, Benjamin/Cummings, Menlo Park.

[17] Deutsch, M. S. and Willis, R. R., (1988) Software Quality Engineering: A Total Technical and Management Approach, Prentice-Hall, Inc., NJ.

[18] Lai, S. T. and Yang, C. C. (1998), "A Software Metric Combination Model for Software Reuse," Proc. of 1998 Asia-Pacific Software Engineering Conference (APSEC'98), Dec., pp. 70-77

Authors

Sen-Tarng Lai received his master from National Chiao Tung University, Taiwan in 1984 and PhD from National Taiwan University of Science and Technology, Taiwan in 1997. His research interests include software security, software reuse, software quality, and maintenance. He is currently an assistant professor in the Department of Information Technology and Management at Shin Chien University, Taipei, Taiwan.