Deployment Strategy for Secure Systems in Populations of Low Digital Literacy

Juan C Lavariega, PhD[1], Emanuel Casas, MS[1], Lorena G Gomez, PhD[1], Kim Mallaliu, Ph.D[2]
[1] Tecnologico de Monterrey, Mexico, lavariega@itesm.mx, emmanuelcb@gmail.com, lgomez@itesm.mx
[2]University of West Indies,kim.mallaliu@sta.uwi.edu

Abstract*– Nowadays, we depend on multiple computer systems and software applications, which ease our daily lives. From financial and corporate applications to health records and personal apps for keeping records of our daily habits such as food intake, exercise routings or just chatting with distant family members and friends. Undoubtedly, all software applications must exhibit security as one of their top quality attributes, in some applications security is the most valuable characteristics, for example, online financial applications and electronic health records deal with sensitive, private and confidential information. In this paper, we present our software development framework for achieving secure applications. We have used this framework in the construction of a financial application that manages on-line wireless transactions in rural communities and we have starting using the framework in our remote health monitoring and EHR systems. Our framework for building secure applications consists of principles, strategies, and tasks associated to the software development process.

***Keywords*— Secure software, software development, design, human factors.**

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2015.1.1.230
ISBN: 13 978-0-9822896-8-6
ISSN: 2414-6668

Deployment Strategy for Secure Systems in Populations of Low Digital Literacy

Juan C Lavariega[1], PhD, Emanuel Casas[2], MS, Lorena G Gomez[3], PhD, Kim Mallaliu[4], Ph.D
[1,2,3]Tecnologico de Monterrey, Mexico [1]*lavariega@itesm.mx,* [2]*emmanuelcb@gmail.com* [3]*lgomez@itesm.mx*
[4]University of West Indies,*kim.mallaliu@sta.uwi.edu*

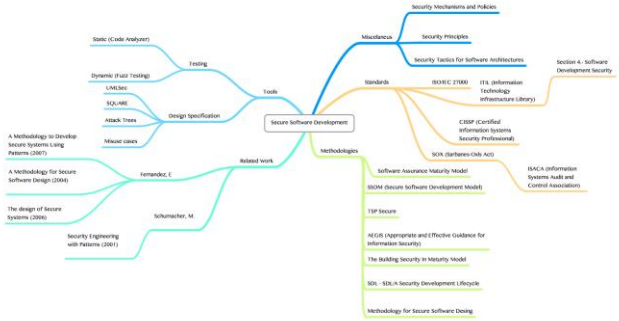
ABSTRACT
Nowadays, we depend on multiple computer systems and software applications, which ease our daily lives. From financial and corporate applications to health records and personal apps for keeping records of our daily habits such as food intake, exercise routings or just chatting with distant family members and friends. Undoubtedly, all software applications must exhibit security as one of their top quality attributes, in some applications security is the most valuable characteristics, for example, online financial applications and electronic health records deal with sensitive, private and confidential information. In this paper, we present our software development framework for achieving secure applications. We have used this framework in the construction of a financial application that manages on-line wireless transactions in rural communities and we have starting using the framework in our remote health monitoring and EHR systems. Our framework for building secure applications consists of principles, strategies, and tasks associated to the software development process

Categories and Subject Descriptors
K.6.1[**Project and People Management**] Strategic Information Systems Planning

General Terms
Design, Secure Software, Human Factors.

Keywords
Secure software, software development.

1. INTRODUCTION
Multiple software systems ease our daily lives, from financial and corporate applications to health records and personal-mobile apps. As the adoption, reach and functionality of software applications continue to grow, software developers are increasingly required to integrate security features in their development lifecycles. At the same time, the low entry barrier for software production has expanded the development community beyond the tradition of thematic experts to include vibrant and productive entrepreneurs with practical, rather than strictly academic, backgrounds. Without exception, all developers require to integrate security features into their software products to reduce users' vulnerability to malware and loss or theft of their private information. An approach, useful for experts and novice developers, to navigate to the plethora of security assessment alternatives (methodologies, tools, standards, etc) is the use of guidelines. Figure 1, illustrates the relationship of secure software development alternatives that we have reviewed.



This paper presents our framework for supporting the implementation of secure software based on well-established principles. The paper describes the security principles as the foundation for secure software development, categorizes them according to their relationship to key aspects of the development process, identifies specific tasks, which we use to put the principles into practice, and describes the framework in terms of the mappings between principles and tasks in the context of the software development lifecycle.

Even though our framework is still a work in progress, we have tested its applicability in the development of a mobile application for financial transaction in underdeveloped communities, and currently we are applying it for our projects in remote heath monitoring in rural areas. In this paper, we present our framework to guide, track and appraise the development of secure software.

We organize the remaining of the paper as follows. Section 2 addresses the concept of security and its relevance to software systems. It also presents related preliminary work in defining security principles for software development..

Section 3 presents our framework for secure software development. Section 4 shows how we applied the framework in the construction of the mobile financial transactions application. Section 5 concludes our paper and addresses future work.

2. SECURITY IN SOFTWARE

Security is the quality attribute of software systems to protect against accidental or deliberate intrusion. Security implies the protection of that any information asset from unauthorized access while still providing full services to authorized users and systems.

The low entry barrier for software production has expanded the development community beyond the tradition of thematic experts to include entrepreneurs with practical, rather than strictly academic, backgrounds. At the same time, the adoption, reach and functionality of software applications continue to grow, and all software developers are increasingly required to integrate security features into their development lifecycle

Security principles are the basis for deriving system requirements and they are applicable to all stages of the software development lifecycle. They reduce design flaws that directly affect application security. They are architecturally neutral and programming language independent.

In [1] J.H.Saltzer published seven principles for secure system design as lessons learned regarding data protection in the implementation of one of the first timeshared operating systems. In [2] Saltzer and Schroeder published eight systems design principles that contribute to the flawless implementation of security systems using the notion of security principles as a reference 'mechanism' to appraise the security of integrity strategies. Additional security principles appear on [15, 14, 21, 19, 20, 16, 7, 18]. Each of those authors provides between seven and thirty-five principles.

We have analyzed the aggregate set of principles from all of the sources to eliminate redundancies in naming as well as objectives. The result is a reduced set of twenty-eight orthogonal principles that we will present in the following section as part of our framework for secure software development.

We also included in our research a survey of software development methodologies, because several methodologies have been defined for integrating security throughout the life cycle of software development, as in [14] where the authors assert that integral to secure software are the processes of designing, building and testing

3. FRAMEWORK FOR GUIDE TRACK AND APPRAISE THE DEVELOPMENT OF SECURE SOFTWARE

Our framework relates the security principles to specific tasks performed at several phases of the software development cycle. As a guide to ensuring that security is integrated into the implementation of the software development life cycle, it is useful to categorize security principles according to their relationship to: security goals, policies and design philosophy; requirements gathering; architectural design; security mechanisms; and recovery from failure. The general list of security principles was taken from the most significant sources of existing security principles. The amount of security principles presented by each of the sources varies from seven to thirty-two principles. The list presented in this research, shows in compendium different principles that completely differs in the sources.

We classified security principles into the following categories security goals, policies and design philosophy; requirements gathering; architectural design; security mechanisms; and recovery from failure.

3.1 Classification of Security Principles

Security Goals, Policies and Design Philosophy:

P1. Define product security goals and action items to achieve them [7, 19]
P2. Establish a sound security policy as the foundation for design, considering security as a product feature [7, 19]
P3. Define Secure Defaults [7, 19]
P4. Design the security process as an integral part of the overall system design [7].

Requirements Gathering:

P5. Assume nothing while gathering requirements [20]
P6. Use common language in developing security requirements [19].

Architectural Design:

P7. Strive for simplicity and for operational ease of use [20, 21, 14, 19]
P8 Implement layered security [7]
P9 Ensure no single point of vulnerability [7]
P10 Practice defense in depth [14, 21, 20, 7]
P11 Take for granted that external systems are insecure [21, 19, 7]
P12 Limit or contain vulnerabilities, implementing Sandboxing or Compartmentalizing [19, 20, 7]
P13 Minimize the system elements to be trusted [14]
P14 Minimize Attack Surface Area [21]

P15 Delineate the physical and logical security boundaries governed by associated security policies [7]
P16 Use boundary mechanisms to separate computing systems and network infrastructures [7]
P17 Identify and secure the weakest link [20].

Security Mechanisms:
P18. Implement Separation of Duties and Least Privilege [14, 21, 19, 20, 16, 7].
P19. Do not implement unnecessary security mechanisms [7]
P20. Authenticate users and processes to ensure appropriate access control decisions both within and across domains [14, 19]
P21. Separate critical from noncritical information [7]
P22. Use unique identities to ensure accountability [7]
P23. Do not trusts Security through Obscurity [14, 21, 7]
P24. Protect information while being processed, in transit, and in storage [14, 19]
P25. Protect against all likely classes of attacks [7].

Recovery from Failure:
P26. Develop and exercise contingency or disaster recovery procedures to ensure appropriate availability [7]
P27. Fail and recover securely [21,20,7]
P28. Design and implement audit mechanisms to detect unauthorized use and to support incident investigations [19].

3.2 Classification of Security Related Tasks in Software Development.

A primary objective of any software development process focused on information security is to produce secure software by reducing the possibility that designers and developers introduce vulnerabilities in design and coding [7]. Although these phases are critical to the protection against vulnerabilities in software systems, they are not the only stages of vulnerability. Indeed, Khan & Zulkernine [8] assert that integral to safety software are the processes of designing, building and testing software. Several methodologies have been defined for integrating software engineering security throughout the life cycle of software development.

The methodologies includes in our study were: Secure software development lifecycle (SSDLC); Structured Systems Analysis and Design (SSDM); Team Software Process for Secure Software Development (TSPSecure); ISO/IEC 10227, Security Development Lifecycle (SDL); Microsoft Secure Software Development (MSSD); Building Security In Maturity Model (BSIMM); Simple Service Discovery Protocol (SSDP); Apvrille; Secure Software Development Process Model (S2Dprom); and, Software Evaluation Framework SEF. Table 1 compares these methodologies in terms of their coverage of tasks relating to training, planning, analysis, design, design review, coding, code review, testing, deployment, maintenance, feedback and assurance.

Table 1 Comparison between Applicable SSDLCs

Software Development Lifecycle Phase / Process or Methodology	Training	Planning	Analysis	Design	Design review	Coding	Code review	Testing	Deployment	Maintenance	Feedback	Assurance
SSDM[8]		*		*		*		*	*	*		
TSPSecure[9]		*		*	*	*	*	*	*			
ISO/IEC 10227[10]		*		*		*		*				
SDL[11]	*	*		*		*						
MSSD[12]		*	*	*		*						
BSIMM[13]		*		*		*		*			*	
SSDP[14]		*		*		*						
Apvrille[15]		*		*		*		*		*		
S2DProm[16]		*	*	*		*						*
SEF[17]	*	*	*			*		*	*		*	

We extracted the following tasks as representative for each phase in the software development lifecycle. These tasks are universally applicable to the methodologies under consideration:

Planning:
T1. Product security objectives definition [18]
T2. Product Development strategy specification [3, 18]
T3. Definition of security requirements [2, 3, 4, 18]
    a functional
    b. non-functional
T4. Prioritization of requirements [8]
T5. Assessment of requirements [8]

Analysis:
T6. Definition of security use cases [4, 8, 13]
T7. Definition of abuse cases [2]
T8. Proposed test cases [4, 8]
T9. Risk analysis in security and privacy [1, 2, 3, 6, 8, 18]
T10. Threat modelling [6, 8, 18]
T11. Definition of mitigation plans [3, 8, 13]
T12. Identification of roles of users [8]
T13. Identification and categorization of vulnerability [3, 8]
T14. Analysis of the attack surface [6, 8]

Design:
T15. Functional design specification [6, 8]
T16. Reinforcement of rules through the proposed architecture [4, 5, 8]
T17. Design decisions to mitigate threats [5, 8]
T18. Prioritization of design decisions [8, 2, 4]
T19. Evaluation of safety in design [8, 13]

T20. Identification of resources and trust boundaries [8]

Coding:

T21. Select language, development platforms and components. Keep in mind the need to remove security flaws and prevent their initial insertion. [1, 6, 8, 18]

T22. Monitoring of standards and guides for secure coding [3, 8]

Training:

T23. Elaboration of user training, education and awareness programs

Deployment, Maintenance and Feedback:

T24. Design of plan of incidents and reports
T25. Design of operation and maintenance manuals
T26. Definition of safety management procedures
T27. Monitoring of security
T28. Security update

Testing and Assurance:

T29. Run tests:
- a. Penetration
- b. Unit
- c. Functional

T30. Definition of strategy and testing tools
T31. Dynamic analysis
T32. Fuzz testing
T33. Execution of risk-based security tests
T34. Execution of vulnerability assessment

Reviews:

T35. Specification of security patches
T36. Elaboration of strategy for design review
T37. Elaboration of strategy for code review
T38. Elaboration of strategy for security inspections
T39. Verification and testing of security

3.3 Framework Discussion

Our framework maps all security related task to the relevant security principles. The framework performs the mapping in the context of the development lifecycle therefore producing simple reference aids for secure software implementation.

In our framework, we partition the overall software development lifecycle into three major segments: inception, development and delivery, as shown in Figure 2. We execute planning, analysis and design during inception; coding during development; and, deployment and training during delivery. At the end of the inception phase, we generate work elements specifications. In addition to the key activities executed during inception, development and delivery, the product under development is reviewed at a number of critical stages; and testing is performed iteratively until the product achieves required functional and performance standards. Review activities span inception and development while testing activities span development and delivery. We decompose the scheme of work for the development phase into features; and we decomposer features into tasks. The review process is iterative and a feature is finished once it includes all quality standards.

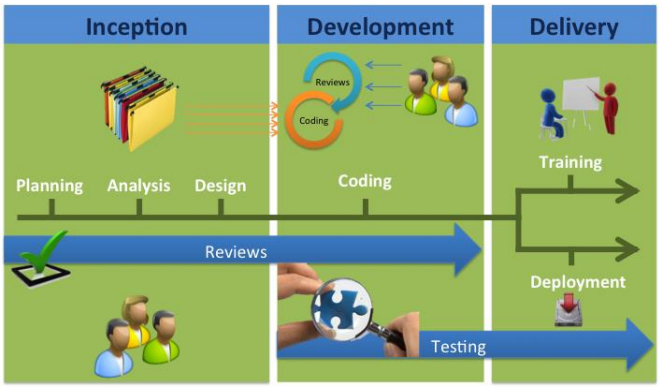


Figure 2. Secure Software Development Overall Strategy

We relate principles and task to contextualize security principles into a secure software development process. Security principles define the objectives to achieve in the software and the related tasks are the activities required to achieve the objectives.

Moreover, we integrate the association between principles and tasks with each phase in the software development lifecycle as illustrated in Figure 3.

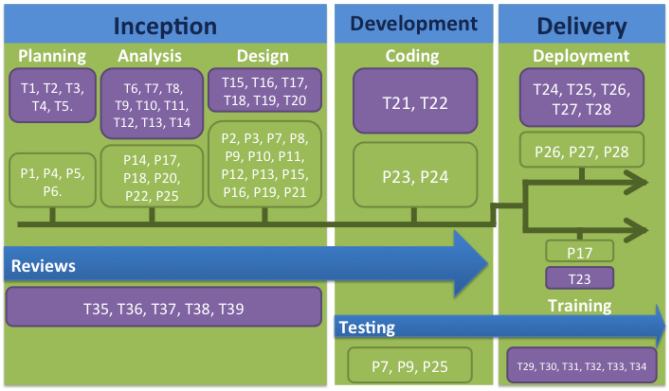


Figure 3. Framework for Secure Application Development.

Figure 3, captures in a single graphic, the primary instrument used in our Framework for Principle-Based Secure Software Development. Developers may consult the Framework as a quick reference guide to ensure adequate treatment of principles and associated tasks in each phase; and to ensure the explicit integration of security principles in the software development process.

4. A TESTBED FOR THE FRAMEWORK

We conceived a mobile application with in-built security and usability, applicable to a wide range of user profiles. Our application called viwi-cash or wvc (short for wireless virtual cash) design enables authorized agencies to provide financial services for clients without formal banking infrastructure. In wvc operations, clients can make deposits, withdrawals, sales, purchases and transfers as well as check their balance and transaction details. Agent administrators manage agency operation and establish all relevant parameters. Agent administrators authorize specific agents to operate the agency, are responsible for wvc accountability reporting and for all activities that directly involve clients. They create clients' wvc accounts; and they create and authorize all money transactions that involve the agency and the client. The wvc system administrator is responsible for configuring the mobile money system and managing user roles and privileges. The wvc system administrator also establishes the security parameters of transactions and verifies account balances. All wvc users, administrators and agent administrators, are registered with the application. Figure 4, shows the mobile money entity interaction model described above.

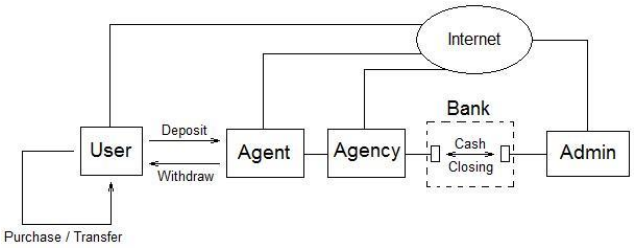


Figure 4 . Wivi Cash entity interaction model

Figure 5 illustrates the general wvc architecture with particular reference to the points of vulnerability: the user, mobile device, communications channel and server. User behavior is the derived vulnerability from the interaction between the user and the mobile device. Malicious code ("malware") and application design vulnerabilities present direct as well as inherited security risks on the mobile device. The wireless communications channel presents ingress opportunities for security breaches. Security threats related to the wvc server arise from information tampering in the form of modification, theft and fabrication.

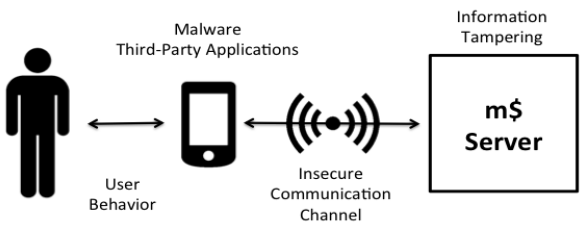


Figure 5. wvc Architecture from a security perspective.

wvc manages highly critical information and is subject to direct economic losses in the event of unauthorized modification, misuse or incorrect administration. The implementation of security mechanisms in application development is therefore essential. Following is an account of the process applied according to the phases stated in our framework: planning, analysis, design, coding, training, deployment and testing (see Figure 3).

4.1 Planning

One practice that causes major security vulnerabilities is the implementation of security features in the final stages of the development process. An associated practice that is fraught with problems is the expectation that the development team will implement security attributes after the team completes the functional requirements. Executing financial transactions is particularly vulnerable to security breaches and the consequences of such breaches can be catastrophic. The most important security objective in wvc is therefore to reduce to a minimum any risks of unauthorized or malicious modification of system data (T1, P1).

The planning process articulates security policies and establishes roles and functions for each member of the development team. These policies, along with our framework, defined a security strategy for the development of the wvc system (T2, P4). We consulted with a board range of stakeholders to ensure nothing was left to assumption while defining the requirements. Stakeholders gave the final approvals (T5, P5).

Broad stakeholder engagement also ensured that the language used for developing the security requirements was simple and fit for purpose (P6). Once the requirements were defined and approved, we prioritized them according to security criticality levels, the most critical being given the highest priority.

4.2 Analysis

We identify and analyze risks (T9) from the definition of security use cases and abuse cases of the system (T6, T7). Similarly, we identify the classes of attacks to which the system is susceptible (P25). Then we defined test cases to appraise system performance with respect to specific attacks (T8).

Once we identified the classes, we defined and categorized vulnerabilities (T13). This activity was useful in modeling system threats (T10), consequently we proposed strategies for preventing attacks (T11). Having done this, we identified the entire surface area of attack (T14), and we proposed strategies for protection and attack minimization (P14).

One of the critical issues of the analysis phase is that of security problems resulting from user interaction with the

system. It is essential to articulate the roles of users and define the manner in which they will interact with the system. For each role in wvc, capabilities and authorizations were defined to ensure that each user was assigned the least amount of privileges to perform her/his tasks within the system (T12, P18)

4.3 Design

For the functional design specification (T15), is important to define security policies that take into account security attributes and characteristics of the product (P2). The most important policies identified in this point were related to the formation of secure passwords, characteristics of the user ID, login attempts and procedures to subscribe and unsubscribe to wvc.

Through field investigations with a sample user base, we established that one of the most pressing security concerns, for prospective wvc users, is the potential loss of money. For the protection of user information, it was essential to establish a configurable set of security parameters. Administrators set the values for the security parameters depending on the needs of their agencies (P3).

User authentication is the most critical information for design specification of wvc. In accordance with best practice (P21), we separate user authentication from user related information such as personnel information and transaction history. Then we conducted vulnerabilities analysis and threats analysis to implement only the mechanisms necessary and sufficient to prevent successful execution of each type of attack.

We implemented validation of entities by digital certificates and challenge-response passwords, because in wvc communications are performed using unsecure channels. An easy access to mobile application requires a double-factor validation: username and password; and PIN.

We consulted with users at all stages during user interface design to maintain simplicity and operational system ease of use (P7). We presented a feedback strategy incorporating a prototype to prospective users and, based on the interaction with the interface, we obtained valuable feedback about the usability of the system. This feedback was important to the final implementation version. Each time the user performs any transaction that involves money balance modification, wvc shows appropriate messages with all the information related with the transaction. Once the user accepts, $m shows a summary message stating the new balance state.

We developed a software architecture to meet all the considerations and reinforce all rules established in the previous phases of design (T16). Examples of security decisions taken for the design (T17) were as follows:

We used a layered architecture to avoid a unique vulnerability point (P8,P9). Grouping architecture components and layers, depending on the type of services that each component provides, allowed for overall system compartmentalization (P12). Furthermore, we placed one of the most valuable system assets (i.e. information) in just one layer. The layer requires several validations prior grant access to information.

We implemented relevant security mechanism (P10) for each of the layers in the architecture. These mechanisms included malformed or invalid information blocking in input data; and granting access only to authorized users and devices. Once we establish the design decisions, we prioritized them (T18) and evaluated them (T19).

In order to identify the resources and trust boundaries (T20) it was essential to identify the boundaries between the logical components of the system and the infrastructure and to separate them (P16). We defined element-wise security policies (P15) to govern the interaction between each one of the elements already separated. With these policies in place, we were able to identify the trust elements, and minimize them (P13) by categorizing external systems as insecure (P11).

Similarly, for the information resources, it is important to separate critical information from non-critical (P21) information. We proposed different mechanisms for data protection (P24) and we use unique identifiers to facilitate the data usage audit by each entity in the system (P22).

4.4 Coding

The wvc implementation assumed that the code would be broadly available for public scrutiny to avoid trusting that security mechanisms were hidden in the code (P23) and generally as a tool to ensure that coding does not inject errors that lead to security vulnerabilities (T21). We monitored coding (T22) to avoid defect propagation to later stages and we performed revisions as necessary.

We made provisions to prevent the system from going to a vulnerability state. Provisions include give feedback to the user under any system failure, so he or she is able to make an informed choice of actions (P27).

4.5 Training

The weakest link in the security environment is the user (P17); therefore, we developed a comprehensive strategy for end-user training (T23). We focus the training in knowledge

and skills necessary to practice secure behavior. The training treated with preventative as well as reactionary measures relating to a range of vulnerability exploitations. In the fishery community, training was particularly significant due to the low digital literacy of the inhabitants. We will report our experience in community intervention in other paper.

4.6 Development, Maintenance and Feedback

After software implementation, it is important to document contingency action strategies. We created failover procedures (P26); incidents plans (T24); operations and maintenance instructions (T25); and administration manuals (T26) to ensure appropriate levels of system availability. We made use of transaction logs, which stored the activities of all users in the system (T27) as a means to support security-auditing mechanisms (p28)

4.7 Testing and Quality Assurance.

The results obtained from the analysis phase are important to carry out the testing phase. In the analysis phase of wvc, test cases were defined, a risk analysis was performed and the types of possible attacks were identified (P25). Then, in the testing phase a vulnerabilities verification strategy (T34) was designed to test execution (T30) based on the risks identified (T33).

We performed dynamic analysis (T31) on the prototype of the system before its release. We also performed unit and functional testing (T29) at the beginning of the testing phase. Other types of testing that we conducted on the fully functional prototype were penetration testing (T29) and fuzz testing (T32)

4.8 Reviews

Reviews play an important role in the implementation of the security features of the system during the initial phases of the development project. For inception and development stages, it is essential to define a strategy for security inspections (T38). For the coding phase of wvc specifically, we performed code reviews (T37). Based on the results of the reviews we specify changes due to potential vulnerabilities (T35). We also conducted security checks on each feature already implemented (T39).

For the Design phase, we design a review strategy to assess compliance with the elements defined in the inception stage (T36)

5. CONCLUSIONS

Information Technology and especially mobile computing allow us to reduce the digital divide between modern societies and under development communities. However, the increasing sophistication of mobile services is a potential minefield of security vulnerabilities, particularly amongst users with low digital literacy. In this paper, we presented a framework for development of secure applications. In particular we showed how the framework was applied during the development of mobile money, an application that brings on line financial services to under development communities.

We consider that our framework is not exclusive for mobile financial online applications like $m, but also our framework is suitable for any kind of software product requiring security as key property.

This project is the resulting effort of two teams working under a grant provided by the Latin America and Caribbean Collaborative Information and Communication Technologies Research (LACCIR) federation: University of West Indies (UWI) from Trinidad and Tobago; and Tecnologico de Monterrey (ITESM) from Mexico. Even though both teams worked side by side during the construction of $m, the ITESM group focused on the definition of the framework for secure software development, and the UWI focused on applying a multi-disciplinary strategy following our secure software development framework in a low digital literacy community.

6. REFERENCES

[1] Allan, D., Hahn, T., Szakal, A., Whitmore, J., and Buecker, A. 2010. Security in Development: *The IBM Secure Engineering Framework.* Red books.

[2] Chess, B., & Arkin, B. 2011. Software Security in Practice. *IEEE Security & Privacy Magazine*, 9(2), 89–92. doi:10.1109/MSP.2011.40

[3] Essafi, M., Labed, L., and Ghezala, H. Ben. 2007. S2D-ProM: A Strategy Oriented Process Model for Secure Software Development. *International Conference on Software Engineering Advances (ICSEA 2007), (Icsea)*, 24–24. doi:10.1109/ICSEA.2007.59

[4] Fernandez, EB. 2004. A methodology for secure software design. *Proceedings of the 2004 Intl. Symposium on Web*, 21–24-

[5] Horie, D., Kasahara, T., Goto, Y., & Cheng, J. 2009. A New Model of Software Life Cycle Processes for Consistent Design, Development, Management, and Maintenance of Secure Information Systems. *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, 897–902. doi:10.1109/ICIS.2009.175

[6] Howard, M., & Lipner, S. 2006. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software* (p. 304). Redmond, Washington: Microsoft Press.

[7] Howard, M., and LeBlanc, D. 2002. *Writing secure code* (2nd ed., p. 800). Microsoft Press

[8] Khan, M. U. A., & Zulkernine, M. 2009. On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software. *2009 33rd Annual IEEE International Computer Software and Applications Conference*, 353–358. doi:10.1109/COMPSAC.2009.206

[9] Mallalieu, K., 2012. *Networks for Development: Caribbean ICT Research Programme, Trinidad and Tobago*. Technica Report. University of West Indies at St Augustine. May 1, 2012

[10] Mallalieu, K. & Sankarsingh, C. 2012. *Contemplating Mobile Applications for Small-Scale Fisheries in Trinidad and Tobago.* In Dunn, H. (Ed.) Ringtones of Opportunity: Policy, Technology and Access in Caribbean Communications. Kingston: Ian Randle Publishers.

[11] Mallalieu, K., and Sankarsingh, C. 2012. mFisheries: Lessons in First Cycle Design of a Context-appropriate Mobile Application Suite. *International Journal of Technology and Inclusive Education,* 1(1), 9-16.

[12] Mohammed, E., Ferreira, L., Soomai, S., Martin, L. and Chan A. Shing, C. 2011. Coastal fisheries of Trinidad and Tobago. *In S. Salas, R. Chuenpagdee, A. Charles and J.C. Seijo (eds). Coastal fisheries of Latin America and the Caribbean*. FAO Fisheries and Aquaculture Technical Paper, 544, 315–356.

[13] Nunes, F. J. B., Belchior, A. D., and Albuquerque, A. B. 2010. Security Engineering Approach to Support Software Security. *2010 6th World Congress on Services,* 48–55. doi:10.1109/SERVICES.2010.37

[14] Saltzer, J H, and Schroeder, M. D. 1975. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278–1308. doi:10.1109/PROC.1975.9939

[15] Saltzer, J. H. 1974. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7), 388–402. doi:10.1145/361011.361067

[16] Simpson, E. S., Howard, M., Corp, M., and Randolph, K. 2011. *Fundamental Practices for Secure Software Development 2ND EDITION A Guide to the Most Effective Secure Development Practices in Use Today* (p. 56).

[17] Sodiya, A. S., Onashoga, S. A., & Ajayi, O. B. 2006. *Towards Building Secure Software Systems,* 3(2000).

[18] Software Engineering Institute. *2010. TSP-Secure. CERT, Carnegie Mellon University*. Retrieved from http://www.cert.org/secure-coding/secure.html

[19] Stoneburner, G., Hayden, C., & Feringa, A. 2004. *Engineering Principles for Information Technology Security*. Retrieved from http://csrc.nist.gov/publications/nistpubs/80027A/SP80027RevA.pdf

[20] Viega, J., and McGraw, G. 2002. *Building secure software*. Boston: AddisonWesley.

[21] Wiesmann, A., Stock, A. van der, Curphey, M., and Stirbei, R. 2005. A guide to building secure web applications and web services. *The Open Web Application* . Retrieved from https://www.owasp.org/images/b/b2/OWASP_Development_Guide_2.0.1_Spanish.pdf