Improving Software Security in ExtremeProgramming Methodology.

Ahmad A. Ababneh
Computer Science Department
Jordan University of Science and Technology
aaababneh150@cit.just.edu.jo

Mohammad A. AL-Rudaini
Computer Science Department
Jordan University of Science and Technology
maalrudaini15@cit.just.edu.jo

Mohammed A. Khasawneh
Concordia Institute for InformationSystems Engineering
Concordia University
moh_kh86@yahoo.ca

Fawaz A. Khasawneh
National University College of Technology (NUCT)
fkhasawn@nuct.edu.jo

Ammar Abdallah
Software and IT Engineering, École de Technologie Supérieure, University of Québec Montréal, Canada
ammar.abdallah.1@ens.etsmtl.ca

ABSTRACT

Agile software development has been used by business to produce a more adaptable and simple software development process, i.e. making it possible to develop software at a faster rate and with more agility during development. There are however concerns that the higher complexity, refactoring rate, and lack of documentation are creating less secure software. Software Security in other hand is one of the most important factors in software projects success, but it costs more time and effort in the software development life cycle and increases the overall complexity of the software project. In this paper, the authors suggest some methodologies that may enhance the overall software security by implementing them into extreme programming (XP) life cycle.

Keywords
Extreme Programming, Pair Programming, Agile software development, Security Education, Threat Modeling

1. INTRODUCTION

Security is an important aspect that should be focused on. This can be accomplished on both data in transit and data at rest. Encryption is an essential component to implement this Saleem et al. (2014) Taha et al. (2020). Software Engineering Methodologies are always facing an immense amount of obstructions and failures in all phases of Software Development Life Cycle (SDLC). Agile development methods are usually used to iteratively create the software systems and they can easily handle ever-changing business requirements Beck et al. (2001).

1.1 Agile Software Development

Agile Software Development manifesto Beck et al. (2001) is detailed through a number of common principles for agile processes, putting customer satisfaction as the highest priority through early and continuous delivery of valuable software in sustainable development, easing the communication as business people and developers must work together daily throughout the project, evaluating their effectiveness at regular intervals and adjust their behavior accordingly, using a face-to-face meetings as an efficient and effective method to share and express information, with a high latency response dealing with changing requirements, even late in development, including iterative delivery of working software, as the working software is primary measure of progress, using motivated individuals with the environment and support they need, entrusting them to get the job done, and to self-organize the team which emerge the best and simple architectures, requirements, designs, and continuous attention to technical excellence and good design enhances agility.

1.2 limitations of Agile Software Development

There are some limitations of Agile Software Development Turk et al. (2014) like limited support for distributed development environments, subcontracting, building reusable artifacts, software development environment involving large teams, developing safety-critical software, and developing large and/or complex software.

1.3 Extreme programming

Extreme programming (XP) Beck (2000) Pressman (2005) is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it recommends frequent releases in short-timed development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

Epstein (2008) said "Agile processes are increasingly important in the world of software development, especially in this age of Net-oriented software. An important issue is how agile processes, like extreme Programming and Scrum, can be modified to address important security issues", Ghani and Yasin (2010) done a literature review to determine if XP is compatible with software security engineering or not, Keramati et al. (2008) conducted a logical andmathematical study "to make a balance between costs of decreased level of agility because of security activities and benefits from developing more secure systems".

1.4 Pair Programming

Pair programming is one of XP methodology techniques witch was identified as a good methodology to conduct a real team work collaboration that reduces the headache of code reviews and hence lowering down the refactoring rate, and was tested and studied in many researches like Nawrocki and Wojciechowsk (2000) and Cockburn and Williams (2000) did.

1.5 How to improve software security?

The best way to ensure security of the software product is by improving the security in the early stages of development by adopting security-based tactics, then use software security analysis tools to detect and take measures.

1.6 Black Box, and White Box Testing

Black Box, and White Box Testing McGraw (2006) is not really a security testing methodology rather is a classification of security test regarding the visibility of the test subject to the tester tool or specialist. Black Box (Black Hat) Test is destructive-based test done to the software without knowing its internals, providing an external security viewpoint of the software under test. In the other hand White Box (White Hat) Test is constructive-based test done over a software knowing its internal coding and architecture, provides an internal security viewpoint of the software under test, many security test techniques may fall into one or both classifications.

1.7 Vulnerability Test

Vulnerability Test McGraw(2006) is a rapid and advanced white box security test of possible vulnerabilities, it is very affordable ($100 average), with the ability of automation and fast execution (only minutes) it is considered as a good method for first layer security scan, but still needs a manual check to rule out false positives, and exploits to be done resolving the confirmed positives before starting the test again.

1.8 Penetration Test

Penetration Test McGraw(2006) in other hand is an on-air, manual or automated, test with more accurate and in-depth results, it is executed annually or after any major change as white box or black box test, with the ability to rules out false positives, but its long execution time (1 day to 3 weeks), and higher cost (around $4,000 to $20,000) makes it better used as a final stage securitytest rather than a daily, weekly, or even monthly based test.

2. RELATED WORK

Many of previous researches have been conducted to implement many security engineering techniques into agile development methods, Boström (2006) indicated that Agile software development methods, in particular XP, need and can be adapted to conform to security engineering practices as defined by the SSE-CMM and the Common Criteria, and that Aspect-oriented programming can be a useful technology for the implementation of the security features of Access Control, Confidentiality, and Quality of Service instrumentation, since it improves modularity, and easing the evolution towards instrumented Web Services, finally he indicated that both Agile software development and Aspect-oriented programming are a good combination that reinforces the strengths of each other, since Aspect-oriented programming provides a new tool for reducing the pain of the continuous refactoring that is so emphasized in agile methods, Boström et al. (2006) the research was focused mainly on the security in aspect-oriented development method and its implementation in agile development methods to reduce the possible complexity, and Including a security engineer in the project team, assessing security risk together with the on-site customer, propose security related user stories, pair programming with the development team to ensure correct implementation of security, and to spread security thinking in the project team, and documenting the security engineer's pair programming activities to ensure coverage, and complement with further reviews if necessary to help build an assurance argument, then as a preparation before a security review, documenting the security architecture in order to provide an assurance argument, and to complement pair programming with static verification and automatic policy enforcement if possible, Wäyrynen et al. (2004) all that is based on the analysis of how XP deals with the security activities and requirements stated in the Systems Security Engineering-Capability Maturity Model (SSE-CMM) and the Common Criteria (CC).

With the help of project developers from Ericsson AB, a global company, were asked by D. Baca et al.
to grade security improvement activities with respect to requirement, design, implementation, testing and release within a project using agile development methods. These security activities were chosen from three leading software development processes (Cigatel Touchpoints, Common Criteria, and Microsoft SDL) inorder to compare traditional large scale waterfall development security benefits with a smaller, more flexible, agile software development process, the result was that all three traditional processes scaled badly to the chosen agile security processes because of too high costs or not being enough beneficial for the agile conditions, i.e. time constraints and reiterated increments of the product.

Using another security technique Tomanek and Klima (2011) implemented the definitions of security requirements into the Scrum product backlog, in addition to adapting automated penetration tests in the daily sprints, and the manual penetration test(s) after the final version of software increment is released, and with comparison with Baca and Carlsson (2011) they reduce the time of security implementation by using the automated tests to pair with the daily sprints, but with an increase of cost regarding the usage of automated testing tools, and the increasing of cost and time regarding the usage of an ethical hacker to do the manual test after the release of the software product final iteration.

Compared with Tomanek and Klima (2011) research, Azham et al. (2011) performed a detailed research about the security backlog in the Scrum agile methodology, to deal with the security issues they mentioned as "it is not necessary to start a project with a lengthy, upfront effort to document all requirements", "The prioritize feature that want to be release quickly will in a danger if not analyze it with the threat analysis phase", "the time release has been decided, and any addition in security part, will be hard to manage in time", "all product has been integrated and will be testing for a final. When it come to this phase and the security part will cost a lot of time and money here to maintain".

Li and Cui (2010) used the static analysis methods and tools alone or in combination with dynamic detection methods and tools to ensure the software security, regardless of saying the for getting a better software security some security project-based approaches must be used, they didn't all the software project phases.

Rindell, et al (2015) used a widely used Finnish governments security criteria, named (Vahti), as a basis for evaluation of three approaches to software development for a regulated environment. The selected security framework was Microsoft SDL and methods XP, Scrum, and Kanban. The research objective of this study was to use light-weight DESMET evaluation criteria to analyses the adaptability of agile methods tosecurity development,

and to estimate the cost of security-related tasks, they conducted only in a theoretical framework, shows that agile development is readily adaptable to even the strictest security requirements. Also, the message in the studied literature is clear about certain benefit of employing agile methods to develop security-oriented software as the development of a software in numerous iterations towards the finalized product may actually improve security assurance.

Beznosov and Kruchten (2004) unified security assurance approaches and methods into the agile development practices, classified predictable methods and techniques used for security assurance in regard to their acceptability for agile development, and also proposed ways to accommodate the incompatible techniques. Beznosov (2003) proposed extreme Security Engineering (XSE), as an implementation of extreme Programming (XP) practices to security engineering projects.

Siponen et al. (2005) provided an example of how security methods can be added flawlessly to agile software development methods. they explained the addition of this solution to an agile development method, as Feature Driven Development.

3. METHODOLOGY

The paper authors suggest that simplicity and complexity both has a security or risk limitations, so middle balance is best using simplicity in processes when it gives the best outcome, and complexity in other processes when it gives the best-quality outcome.

3.1 How to make XP more secure?

This is the main question that this paper and the authors is focused on and will try to suggest ideas that could answer it, extreme programming (XP) life cycle does not suggest or include security techniques, the lack of security tools, methodologies, or techniques from XP life cycle creates vulnerable software by affecting the project timeline due to the overall continues rise of refactoring cost and time which will lead to negatively affect customer satisfaction.

The main idea for the project is to minimize the security risk by including some security practices into the extreme programming practices as demonstrated in Figure 1 that will rise the initial cost, time, and effort, but will decrease them efficiently in the future software product iterations, and will ease the refactoring process, increasing the total life time of the software product.

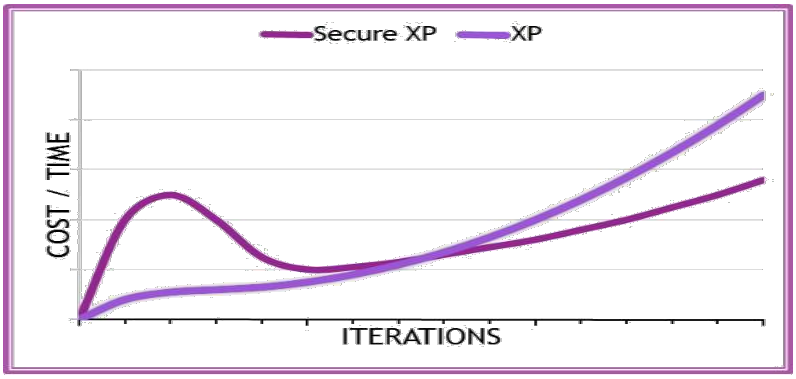

Figure 1: XP Cost and Time with & without Security Enhancements.

McGraw (2006) says that “Software security naturally borrows heavily from software engineering, programming languages, and security engineering. The three pillars of software security are applied risk management, software

security touchpoints, and knowledge. By applying the three pillars in a gradual, evolutionary manner and in equal measure, a reasonable, cost-effective software security program can result".
The cost of discovering technical or security bugs or threats is high in the last stages of the software development life cycle, so the security improvement in software must start at the beginning stages of XP SDLC i.e. the planning and design.

3.2 Security in Planning

The planning is the first phase of XP SDLC, it is a major factor for the success of any software product as well as its security, as it better to handle the security issues and the early stages of the SDLC.

3.2.1 security expert

The involvement of a security expert (ethical hacker) into the project team is a major need specially for large projects, his responsibilities will vary between:

- Educating the entire team and the involved customer personnel about security issues related to the software product.
- Conducting a special training for developers about security related coding and testing weaknesses and techniques.
- Performing security reviews for user stories and customer requirements to identify abuser stories that may occur in the future based on the security requirements identified.
- Identify the data which the application will operate on, and the use-cases for operating on that data that the application will facilitate.
- Specifying the priorities for the security requirements.
- Security tests planning.

This idea will take a higher cost, time, and effort for the XP planning phase, but the identification of the security threats at that level will enhance the following stages, and lower the cost of changing requirements, code review, and refactoring, by using the security requirements, abuser stories, and the security specialist assistance and supervision.

3.3 Security in Design

Design is the second phase of the XP SDLC, it is as important as the planning phase as it is one of the early stages in the SDLC.

3.3.1 security threat modeling

Based on the security requirements and abuser stories, security threat modeling is a good security design methodology often represented as a Data Flow Diagram (DFD) that represents the collected abuser stories as the potential attack points from outside the system, it consists of:

- Identifying the software product assets.
- Creating an architecture overview of the overall software components.
- Decomposing the software product into components each having its own function.

- Model any external dependencies, calls from/to components, and the data store for each use-case as identified in the planning phase.
- Identify and document the expected threats for each component, data flow, and use-case.
- Rating all the threats as well as their related component to identify the security importance of each of them.
- More than one solution must be suggested for each threat to be used in

The threat modeling will be considered as the security reference for any security tool or methodology that will be used in the later stages of the XP SDLC.

3.4 Security in Coding

The coding phase is an important stage in security of the software product, as the most security vulnerabilities a software product may face is related mostly to the coding, XP in other hand emphasize the coding phase by using a programming improvement techniques like pair programming and daily meetings.

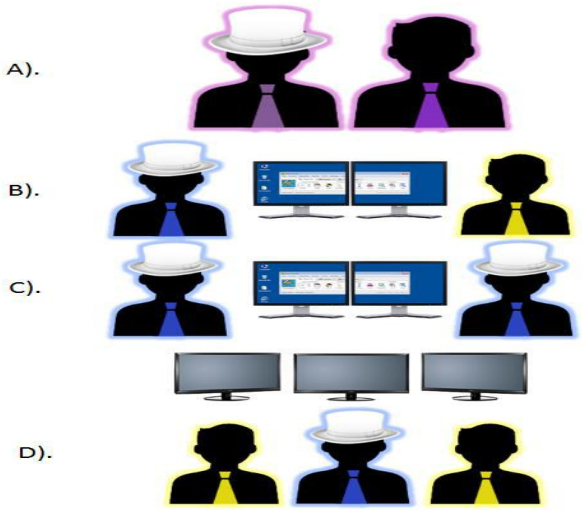

Figure 2: Security Based Pair Programming.

3.4.1 Security Based Pair Programming:

Based on the security education and assistance of the team programmers by the security specialist, the security of the pair programming technique could be enhanced by:

- The pair programming is supervised by both development team leader, and security specialist, i.e. Pair Team Leading, as seen in *Figure* 2(A).
- Pair programming goal must achieve costumer, software, as well as security requirements.

- At least one of the pair programmers is security educated, to ensure the understanding of security requirements of the software unit being programmed, as seen in *Figure*2(B),(C).

- Third security educated programmer may join the pair programming as a security assistance when needed, as seen in *Figure*2(D).

- The variation of security educated pair programmers in the team, as seen in *Figure*2 (B), (C), (D), depends on the security requirement importance, and complexity of the software unit being programmed.

- Using the pair programming session as security education practice, sharing security related programming ideas and information.

- Using the work break time as a security education discussion, sharing up-to-date security related improvements, and threats.

3.5 Security in Testing

Testing and coding phases are interrelated by the fact that the test results are essential for the code refactoring in the next iteration, what we suggest in this phase is to integrate security testing with the code testing as XP has three stages of code testing we suggest the addition of three security tests along with them plus one final pure security test as follows:

3.5.1 Vulnerability Unit Test:

Planned and designed with minimal security test cases (only the important ones), automatically executed after any minor change in code to get less results to be easier to be manually checked, reviewed, then refactored as fast as it won't cost more time over the unit coding process, when the final results of the final execution have been identified to have no true positive treats, mean that the software unit is ready to be integrated.

3.5.2 Vulnerability Integration Test:

Planned and designed with full test cases, automatically executed after each code integration, new results will appear as most results were reviewed at the vulnerability unit test with little more effort to be manually checked, reviewed, and refactored as necessary, when the final results have been identified to have no true positive threats, the software module being programmed is ready to be delivered in the next iteration.

Note that manual check of false positive identified threats, and the manual review and refactoring of the true positive identified threats of the vulnerability unitor integration test, is a good way for a programmer to practice his security education.

3.5.3 White Box Penetration Test:

Planned and designed to achieve the entire security requirements based on the knowledge of the internal design and components of the software product, executed during each iteration acceptance test, the execution is usually automated, or manually after a major change, the penetration test is a realistic tool to be used only in the final stage as it needs longer time, and higher cost, the results are reviewed and any change suggested added to next iteration plan and design phases, the results also logged and compared with the next black box test.

3.5.4 Black Box Penetration Test:

Planned, designed, and executed by an external contracted security specialist (ethical hacker) without having the knowledge of the internal design and components of the software product, the authors suggest that the test is quarterly and manually executed to simulate an external black hat attack. The results of the black box penetration test are logged and compared with the previous white box tests to be used to find any neglected, unknown, or miss-resolved security treats.

3.6 Security after Release

When new iteration is ready to be released, the whole security and technical improvements will be under the test, as the software may face its worst nightmare (failing) if you did not prepare carefully.

3.6.1 Security Monitoring and Tracking:

Any software product needs a monitoring system to sense the change of any situation related to every software module based on the product design diagrams to decide that every module is working properly or not, or they may face a future risk, to deal with it before it becomes a real time risk.

Based on the threat modeling and the security testing results, the authors suggest improving the software monitoring system to deal also with the identification and tracking of any security threat that may happen for the working release.

3.6.2 Heterogeneous Architecture:

Based on the composition of the product into component and multiple solutions for each security risk or even any technical risk, and by the assistance of the security and software monitoring system, the heterogeneous architecture is suggested by the authors to create multiple instances for some components to work as an emergency spare components to ensure the stability of the working software, rules are:

- Applied only on the core or security/technical risk related component.
- Level of heterogeneity is based on the importance of each component.
- Used to handle the unexpected technical errors or security bugs without stopping the components.
- Using multi-instance component, having the same function, but with different construction methods, as it shown in *Figure*3.
- Instances are queued, monitoring the main instance, if it failed, crashed, or identified to have a security bug, then isolated and the next one will take its job instantly or by admin order, as it shown in *Figure*3and *Figure*4.
- Each isolated instance is reviewed, refactored, then returned back to the queue in the next iteration.

As shown in Figure 3 the working software product for example is decomposed into User Interface (UI), Operational (O), Communication (C), and Data Storage (D) Components, the most important technical or security related components named (O2) and (C3) are built as three instances, the lower important components named (UI1), (O1), (D1), and (D3) are built with two instances for each of them, finally the lowest importance components named (UI2)=

\\\\]\], (UI3), (O3), (C1), (C2), and (D2) are built with only one instance as the whole software product can survive with out of them.

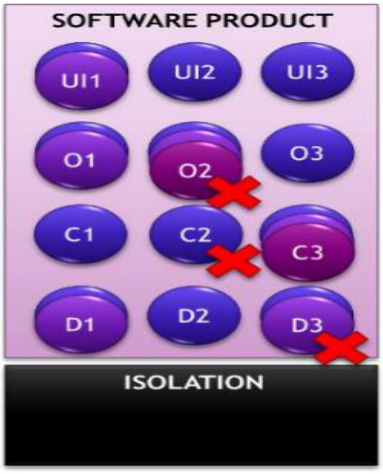

Figure 3: Failed or Bug-Identified Component Instances.

The software components named (O2), (C2), and (D3) sown in Figure 3 are monitored to be in a fault state because of any error, bug, or security threat.

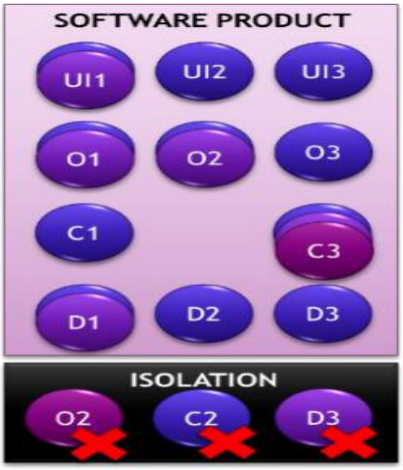

Figure 4: Isolated Instances replacement.

Then as shown in Figure 4 the (O2) fault instance was isolated leaving its function to be done by the next (O2) instance plus one more spare instance, the (D3) fault instance was also isolated leaving only one (D3) instance to handle its function, while the (C2) instance is also isolated but leaving no instance to handle its function as the whole system could survive without that component till next iteration.

Finally, the fault isolated components shown in Figure 4 should be reviewed and refactored to be integrated back into the system in the next iteration.

4. CONCLUSION

Security is an important quality aspect of software systems. Considering security at early stages of SDLC is an important factor dealing with software security issues. In this paper, some security ideas have been proposed to increase the security level in XP, to ultimately improve the security of the software product, by involving a security specialist (ethical hacker) in the whole XP development stages, and by introducing special security education for the software development team programmers and a basic security education for the rest of the team including the involved customer personnel. On the other hand, introducing the security threat modeling as a conceptual view for the expected threat attack points and flow is an important step to implement. Other steps such as introducing the heterogeneous architecture to handle the unexpected technical or security bugs and introducing the security-based pair programming to handle the logical and security code errors, to lower down the security and code review cost and time. Vulnerability testing has been used as security unit and integration test, without increasing the refactoring time. Another test, called white box penetration test which is considered as security acceptance test, to ensure that the security requirements are met. Finally, black box penetration test executed by an external ethical hacker to simulate an external black hat attack, revealing any unexpected or miss-resolved security threat.

5. REFERENCES

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, The agile manifesto, Utah, 2001.

D. Turk, R. B. France and B. Rumpe, "Limitations of Agile Software Processes," CoRR, vol. abs/1409.6600, 2014.

K. Beck, Extreme programming explained: embrace change, Addison-Wesley Professional, 2000.

R. S. Pressman, Software engineering: a practitioner's approach, Palgrave Macmillan, 2005.

R. Epstein, "Getting Students to Think About How Agile Processes can be Made More Secure," in Software Engineering Education and Training, 2008. CSEET '08. IEEE 21st Conference on, 2008.

I. Ghani and I. Yasin, "SOFTWARE SECURITY ENGINEERING IN EXTREME PROGRAMMING METHODOLOGY: A SYSTEMATIC LITERATURE".

H. Keramati and S.-H. Mirian-Hosseinabadi, "Integrating Software Development Security Activities with Agile Methodologies," in Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications, Washington, DC, USA, 2008.

J. Nawrocki and A. Wojciechowski, "Experimental evaluation of pair programming," European Software Control and Metrics (Escom), pp. 99--101, 2001.

A. Cockburn and L. Williams, "The costs and benefits of pair programming," Extreme programming examined, pp. 223-- 247, 2000.

G. McGraw, Software security: building security in, Addison-Wesley Professional, 2006.

G. Boström, Simplifying Development of Secure Software: Aspects and Agile Methods, Department of Computer & Systems Sciences, Stockholm University and The Royal Institute of Technology, 2006.

G. Boström, J. Wäyrynen, M. Bodén, K. Beznosov and P. Kruchten, "Extending XP Practices to Support Security Requirements Engineering," in Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems, New York, NY, USA, 2006.

J. Wäyrynen, M. Bodén and G. Boström, "Security Engineering and eXtreme Programming: An Impossible Marriage?" vol. 3134, Springer Berlin Heidelberg, 2004, pp. 117-128.

D. Baca and B. Carlsson, "Agile Development with Security Engineering Activities," in Proceedings of the 2011 International Conference on Software and Systems Process, New York, NY, USA, 2011.

M. Tomanek and T. Klima, "Penetration Testing in Agile Software Development Projects," arXiv preprint arXiv:1504.00942.

Z. Azham, I. Ghani and N. Ithnin, "Security backlog in Scrum security practices," in Software Engineering (MySEC), 2011 5th Malaysian Conference in, 2011.

P. Li and B. Cui, "A comparative study on software vulnerability static analysis techniques and tools," in Information Theory and Information Security (ICITIS), 2010 IEEE International Conference on, 2010.

K. Rindell, S. Hyrynsalmi and V. Lepp\"{a}nen, "A Comparison of Security Assurance Support of Agile Software Development Methods," in Proceedings of the 16th International Conference on Computer Systems and Technologies, New York, NY, USA, 2015.

K. Beznosov and P. Kruchten, "Towards Agile Security Assurance," in Proceedings of the 2004 Workshop on New Security Paradigms, New York, NY, USA, 2004.

K. Beznosov, "Extreme Security Engineering: On Employing XP Practices to Achieve "Good Enough Security" without Defining It," in First ACM Workshop on Business-Driven Security Engineering (BizSec, 2003.

M. Siponen, R. Baskerville and T. Kuivalainen, "Integrating Security into Agile Development Methods," in System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on, 2005.

N. Saleem, S. Alrabaee, F. A. Khasawneh and M. Khasawneh, "Aggregation function using Homomorphic encryption in participating sensing application," 2014 6th International Conference on Computer Science and Information Technology (CSIT), Amman, 2014, pp. 166-171, doi: 10.1109/CSIT.2014.6805996.

Taha, M.B., Suwi, H., Khaswneh, F., Alzaareer, K. (2020). Adaptive ciphertext policy attribute-based encryption scheme for internet of things devices using decision tree. Revue d'Intelligence Artificielle, Vol. 34, No. 3, pp. 233-241. https://doi.org/10.18280/ria.340301