Software Security for Small Development Teams – A Case Study

Michael Kainerstorfer
Aumayr GmbH
Linzer Straße 46
4221 Steyregg, Austria
+43 732 6440 225
m-kainerstorfer@aumayr.com

Johannes Sametinger
Johannes Kepler University
Altenbergerstraße 69
4040 Linz, Austria
+43 732 2468 9435
johannes.sametinger@jku.at

Andreas Wiesauer
Johannes Kepler University
Altenbergerstraße 69
4040 Linz, Austria
+43 699 11707417
wiesauer@softwaresecurity.at

ABSTRACT

Microsoft is developing wide-spread software solutions like the Windows operating system and the Office suite. In order to improve security of their products, they have introduced the Microsoft Security Development Lifecycle (MS-SDL). Ample documentation about the MS-SDL is available, thus, allowing other companies to adopt the lifecycle as well. We were wondering whether an adoption of the lifecycle is possible and useful for real small development teams, e.g., for a single developing person. In order to find out, we have done a practical test, i.e., we have used the MS-SDL for the development of a small, but real-world software project. The findings will be presented in this paper.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General.

General Terms

Design, Experimentation, Security.

Keywords

Software security, software development, software lifecycle, security development lifecycle.

1. INTRODUCTION

IT security is becoming increasingly important. Software security is an essential aspect thereof. Software bugs and flaws provide the entrance doors for many malicious attacks. The software development lifecycle is crucial for the creation of secure software, i.e. software that is not suffering from such entrance doors. It is not possible to neglect security considerations during development and add security as an additional non-functional requirement just before release. It has to be planned before-hand, because ramifications of this requirement are manifold. Architecture and design of an entire system may have to be adapted in order to guarantee a sufficient level of security and privacy.

Microsoft has introduced security and privacy early and throughout all phases of the software development process. Security-related vulnerabilities in the design, code, and documentation should be minimized and detected as well as eliminated as early as possible. The resulting Microsoft Security Development Lifecycle (MS-SDL) aims at reducing the number and the severity of security vulnerabilities and improving privacy protection. The MS-SDL adds several steps to the software development process and introduces additional roles. These improvements of the development process can be applied incrementally and do not require radical changes of existing development processes. A variant for agile development has also been suggested. However, the MS-SDL appears to be heavy-weight, i.e., made for big development teams that work on millions of lines of code, which is indeed the case for Microsoft teams that, for example, work on Windows and MS Office versions.

It seems natural that smaller development teams will also benefit from the use of the MS-SDL. Developers can embrace lightweight software security practices by using the agile variant of the MS-SDL. We were interested in the following question: Can small and medium-sized businesses (SMEs) with tiny development teams spend the effort of the MS-SDL's additional development steps and benefit from all the additional overhead without getting lost in security issues? Which of these steps make sense and to which extent when developing software, say, as a single person? Which of the suggested roles can be meaningful in such a scenario?

In this paper we will provide a brief introduction to software security, to secure development lifecycles in general and the Microsoft Security Development Lifecycle in particular. We will then report on a small project that had been implemented by use of the MS-SDL. Experiences and lessions learned will conclude the paper.

2. SOFTWARE SECURITY

Software has many quality attributes, e.g., reliability, maintainability, usability or testability. These are non-functional requirements, and software developers often tend to neglect paying sufficient attention to these requirements until (too) late phases in the development process. The same holds for software security, which is more than yet another software quality attribute. Secure software needs to protect data against unauthorized access and to withstand both malicious and inadvertent interference with its operations. Secure software typically uses mechanisms like authentication, access control and encryption.

It is important to understand that it is necessary, but not sufficient to protect computers and computer networks from attack and subsequent intrusion through, for example, firewalls. Firewalls restrict network traffic by deciding what can pass through them based on defined rules. Firewalls have to allow access to Web applications, which would be useless without such access. Malicious input to a web application cannot be detected by firewalls and has to be handled by the application itself.

Secure coding is needed to avoid vulnerabilities like buffer overflows, SQL injection, etc. But security is not just a coding issue. Many design decisions and even the overall system architecture may be affected in order to make an application secure. Therefore, it is grossly inefficient, if not even impossible to develop unsecure software and to make it secure afterwards. Security testing is needed but insufficient.

3. SECURE LIFECYCLE

Software developers have many software lifecycle models to choose from. We can divide between two major paradigms of them. On the one hand, we have so called “heavyweight” lifecycle models that require careful planning, rigorously documented process activities and formalized quality assurance. Examples of such models are traditional waterfall or spiral approaches as well as V-Modell XT. On the other hand, so called “lightweight” or “agile” models exist. They aim at producing software quickly by small iterations which deliver working software that can be presented to customers, who then can demand additional functionality. Therefore, the focus is on the product itself rather than on specification, design or quality assurance artifacts. Examples are eXtreme Programming (XP for short), Crystal or Scrum [1,2,3, 4].

Deciding which process model to choose often is a tedious endeavor, since suitability is influenced by a number of factors, e.g. size and distribution of development teams, stability of requirements, customer participation, personnel qualification or safety criticality [5]. Any of these models can be used in order to develop secure software. But none of them addresses security aspects explicitly.

Software developers need to pay attention to security early in the development process. It is necessary to think about possible threats and mitigation strategies before design decisions are made. For this purpose, threat modeling, which is a structured approach for identifying threats, can be applied [6]. Given a set of possible threats, concrete risks for the system can be evaluated and their mitigation be planned. Therefore, risk management is another important aspect in the security context. Risk evaluation will lead to security requirements which then have to be traced throughout the whole software development lifecycle, e.g. by selecting appropriate architectural styles or design patterns or by implementing security mechanisms. Ideally, security requirements will be foundations for security or penetration test cases, which can verify the correct realization of them. The importance of security to today’s customers and the wealth of activities needed to achieve security suggest an explicit integration of these activities into the software development lifecycle.

Examples for secure lifecycles are the Microsoft Security Development Lifecycle (MS-SDL for short) [7] and the Comprehensive, Lightweight Application Security Process (CLASP for short) from OWASP, the Open Web Application Security Project [8]. Software security touch points are lightweight best practices that are bound to software artifacts rather than a specific lifecycle model [9]. We consider all these examples to be of value for increased security. We have opted for the MS-SDL simply because the prominence and reputation of Microsoft helped in convincing the employer to use the lifecycle.

Even though additional activities have to be done in secure lifecycle, they aim at reducing the total cost of development. Cost reduction is possible, because the costs for fixing vulnerabilities are highest after an application has been deployed already. The National Institute of Standards and Technology (NIST) has estimated that code fixes performed after release can result in tens of times the cost of fixes performed during the design phase [10].

4. MS-SDL

The concepts of the Microsoft Security Development Lifecycle (MS-SDL) were formed with the Trustworthy Computing directive in 2002. At that time, "security pushes" were made to improve security and reliability of existing (Windows) code. MS-SDL aims at assisting developers to build more secure software and to protect end-users.

The MS-SDL was established in 2004 and designed as an integral part of the software development process at Microsoft. Windows Vista was the first operating system to go through the full process. Over the years, MS-SDL has matured into a well-defined methodology, and Microsoft has made guidance papers, tools and training resources available to the public. The MS-SDL consists of several phases, i.e., five core phases, and one pre-phase as well as one post-phase. We will briefly describe these phases in subsequent sections.

Pre-SDL. Software security training of involved people is a prerequisite for the use of the MS-SDL. Microsoft developers, testers, and program managers must attend at least one unique security training class each year. There are many fundamental software security concepts that have to be understood by people who are involved in a software development project.

Requirements. Security and privacy requirements of a software system have to be specified in order to optimize the integration of security and privacy during the development project. This phase consists of three practices: (1) establishing security and privacy requirements, (2) defining quality gates and bug bars, and (3) performing a security and privacy risk assessment.

Design. The security architecture is defined and documented in the design phase. This phase includes (1) the establishment of design requirements, (2) an analysis of the attack surface, and (3) the modeling of threats.

Implementation. Secure software development is ensured in the implementation phase by enforcing security practices, i.e., (1) the use of approved tools, (2) the deprecation of unsecure functions and APIs, and (3) the performance of static analyses.

Verification. Testing the software against security and privacy requirements of phase 1 is done in the verification phase. It consists of (1) dynamic analysis, (2) fuzz testing, and (3) attack surface review.

Release. Preparing software for final release requires an (1) incident response plan, (2) a final security review, and (3) a release archive.

Post-SDL. A response plan is needed with preparations for potential post-release issues. The software development team must remain available to address any possible security issues.

Agile Development. The SDL for agile development integrates security practices into agile software development methodologies. It reorganizes security practices into three categories, i.e., (1) every-sprint practices, (2) bucket practices (to be completed on a regular basis), and (3) one-time practices.

Roles. The MS-SDL includes general criteria and job descriptions for security and privacy roles. These roles are filled during the requirements phase and provide the organizational structure necessary to identify, catalogue, and mitigate security and privacy issues during development.

5. CASE STUDY

For many years, an Austrian-based small-sized business in ventilation engineering had used text messages via email for order processing. Needless to say, there were many drawbacks to that solution. Eventually, the decision was made to develop a web application and to automatically pass on clients' orders to the internal IT infrastructure, e.g., to the production order system. The IT department consisted of three people, one of which was expected to develop the application. Security was important for obvious reasons and the decision was made to take security and privacy serious during development. The requirements were rather clear and as a consequence, sequential development with the MS-SDL was opted for.

Problem Statement. The order system to be developed is rather straight-forward. It encompasses several security-relevant issues. These include the administration of user data (user name, password, company-related information), the transmission of order information over the Internet, and the provision of a public interface to the company's web server connected to the IT infrastructure. The project team for the development of the system was quite minimal. The head of the small IT department served as responsible project manager. Another single person was both the project leader and the entire development team. The project manager and three users of the existing order system made sure that functional requirements were complete and consistent. The three users also served as beta testers to improve usability of the new system. Project consultation was provided by security experts from the local university. Due to the small development team, a security leadership team had not been installed. The responsible project manager had to serve both as security advisor and as security team.

Pre-SDL. Security training of the single software developer consisted of a lecture at the university about software security. The lecture dealt with introductory security topics including security concepts, secure coding, threats and countermeasures, secure software lifecycle. Additional information about software security like [9], information about the MS-SDL like [7], [10], [11], and information about secure coding and web application security like [12] and [13] had been studied in more detail, both as preparation for the project and in parallel with the project.

Requirements. Security and privacy requirements are necessary due to the distributed nature of the system. For example, the volume of orders may be of interest for competitors of both the ventilation company and its clients. While security has been seen as an important issue, the privacy risk for data transmission was assessed to be only modest. This is true because only client IDs and information about the order of or an offer about ventilation items is to be sent over the Internet. Quality gates and bug bars had not been used explicitly. A simple spreadsheet had to serve as a security bug tracking system. The single person project team did not warrant buying one of the quite expensive commercial systems for that purpose.

Design. Various client companies send their orders over the internet to the web server that stores these orders in a queue, see Figure 1. Orders are also stored locally at the clients during creation,

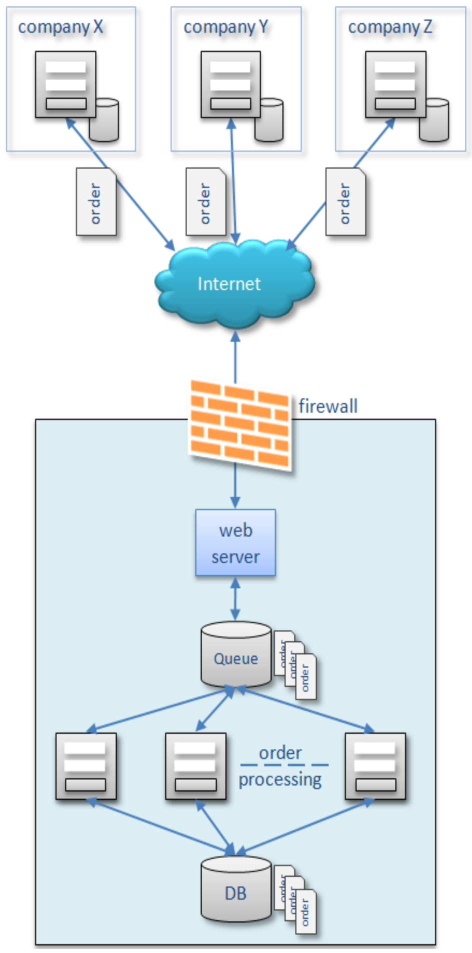

Figure 1: Architecture of Ordering System

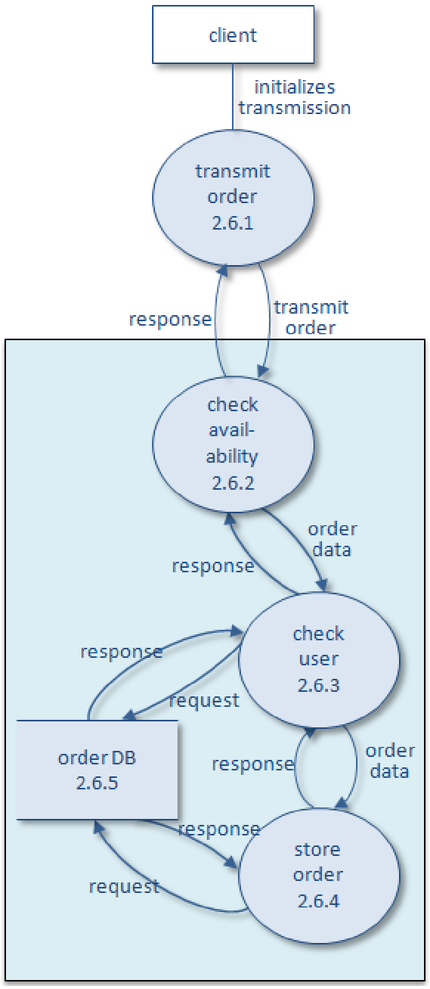

Figure 2: Scenario of detailed order transfer

i.e., before being submitted. Before being accepted, the orders have to be reviewed and schedule by skilled employees and are then stored in the production database.

Threat modeling has to be done at various levels of detail, starting with scenarios of functional requirements and the identification of potential threats. Several main components had been identified in this process, i.e., the client system with a local database that contains order information, a web service that receives orders and stores them on a database on the server, and a timer that checks this database and sends e-mail notifications to people that are responsible for the processing of orders. The entire process had been divided into scenarios that were refined to a level where an identification of risks was possible. An example scenario is given in Figure 2 where the transfer of an order from the client to the database is shown.

Potential threats for the software system were categorized based on the goals and purposes of the attacks, i.e., STRIDE [15]. A total of 21 threats had been identified, see Table 1. Threats 1 through 11 were identified in the scenario "client interaction with ordering system". Threats 12 through 16 belong to the scenario "client transmission of ordering information", and threats 17 through 21 evolved in the scenario "order acceptance". The upper case letters in the second column indicate the STRIDE category. S stands for *spoofing identity*, T for *tampering*, R for *repudiation*, I for *information disclosure*, D for *denial of service*, and E stands for *elevation of privilege* [15].

Table 1: Identified Threats

ID	Threat
T1	Anonymous usage
T2	Manipulation of MK2.ini file
T3	Manipulation of user.ini
T4	Manipulation of license.ini
T5	Manipulation of formparts.ini
T6	Manipulation of database
T7	Denial of reporting of orders
T8	Concurrent access to database
T9	Network failure
T10	Erroneous inputs
T11	SQL injection
T12	Pretention of wrong identity
T13	Manipulation of order data
T14	Denial of order
T15	Eavesdropping
T16	Web server overload
T17	Unauthorized usage
T18	Unauthorized processing
T19	Data manipulation
T20	Denial of order processing
T21	Data disclosure

For example, if a client has completely entered all data of an order, the data will be sent to the web service. First, the availability of the web service will be checked, then authentication information will be sent, and finally, order information will be sent. Several threats are possible in this simple scenario. An order might be sent in the name of a different client (spoofing). Order information might get modified while being transmitted to the server (tampering). The client may later deny to have made an order (repudiation). Information about the order may get into the hands of a third party (information disclosure). Also, the web service may get hit with numerous orders (denial of service). The risks of all these threats had been specified, and appropriate countermeasures had been defined, e.g., the use of secure transmission via the https protocol in order to avoid information disclosure.

The threats were rated with a risk rating between 7 and 10 on a scale between 1 (low risk) and 10 (high risk), see selection in Table 2. Four of the risks were eliminated by the security advisor who assigned lower risk values. The ratings were determined according to DREAD, i.e., by assessing the damage potential, reproducibility, exploitability, affected users, and discoverability [13].All external dependencies had to be identified, e.g., the Windows operating system, the .NET framework, MS SQL Server, a Cisco Systems hardware firewall.

An attack surface analysis is necessary to define all entry points that can be used to interact with the system deliberately, unintentionally or maliciously. We had identified the input of data via the

Table 2: Some Risks and Countermeasures

ID	Risk	Countermeasure
T5	10	Database authentication via password
T6	10	Database authentication via password
T11	9.6	Input validation
T17	9.4	Authentication (user name, password)
T15	9.2	Encryption via HTTPS
T10	9.0	Input validation
T12	9.0	File encryption (User.ini)
T3	8.6	File encryption (User.ini)
T19	8.6	File encryption (User.ini)
R7	8.4	Logging of important activities
T21	8.4	Authentication (user name, password)
T4	8.2	File encryption (license.ini)
T13	8.0	Encryption via HTTPS
T14	8.0	Logging of additional information about ordering user, e.g., Windows user name
T2	7.4	Integrity check of MK2.ini at system launch
T16	7.4	Configuration of web server in IIS console (max. number of requests)
T20	7.0	Logging of important activities

GUI, the manipulation of files being used by the system, i.e., several configuration and database files, and the network traffic. Again, all entrance points got listed, evaluated, and provided with appropriate countermeasures. For example, every kind of user input is validated, external resources, e.g. files and databases, are encrypted and are checked for integrity regularly and data is transferred exclusively via encrypted communication channels.

Implementation. Best practices for development had been followed as much as possible. Naturally, a programmer's security experience level plays a major role in the development of secure software. Therefore, books from leading security experts, e.g. [11] and [13] had been studied prior to implementation. Writing a user manual, systems documentation and a setup manual goes without saying. In addition, any tools that were used had also been described carefully. For example, a simple tool had been developed in order to create configuration files for users to let them authenticate with the order server.

Verification. Fuzz testing, code review and penetration testing had been done by the developer. This is an unperfected approach. It is in contrast to the MS-SDL and should be avoided. In our situation, shortage of manpower did not leave any other choice. Fuzz testing had been applied to all the forms available to users. A small piece of program had been written to randomly fill in all the input fields and then send the form to the server. There were no severe problems emerging from these tests, but input validation had been extended to avoid unusual values that had lead to strange list displays. The configuration file had also been fuzzed. As a result, the software system either did not work at all, e.g., when a wrong working directory had been specified, or some functions did not work, e.g., when specifying wrong path values. As a consequence, the values in the file were checked more thoroughly and precise error messages were provided for the user.

Code reviews were done explicitly, but by the developer. To save time, any findings were immediately corrected in the source code. This again is unperfected, because there are no records of the results of these reviews. In addition, this course of action lacks the benefits of having an external view on the code. Mostly, comments and style were improved. Security-relevant findings were not made in this process.

Penetration testing included a set of load tests in order to make sure that the expected load will not be too high for the system. Additional tests, for example, for SQL injection have also been made. The danger of SQL injections is mitigated by not allowing special characters and SQL key words in any input text. Cross-site-scripting, for example, had not been an issue, as a rich client rather than a web-site had been used. In addition, data uploaded to the server will never be accessed via a Web interface. The system was also used by the three test persons mentioned at the beginning of this section. However, due to a lack of security experience, these testers were mostly concerned about usability rather than about security.

The security push was scheduled to be spread over an entire week with meetings of at most one hour. The security advisor, the development person, and one of the test persons were attending the meetings. All security-related results were reconsidered, i.e., security requirements, threat models, code reviews, tests, documentation. As a consequence, the use of an unqualified certificate for data encryption was endorsed. The attack surface was scrutinized again, but no further attack patterns had emerged as a result. Smaller cosmetic changes were made in the threat model. Penetration testing was found to be rather basic, but the security advisor still voted for a "go", because a manageable number of well-known clients that would use the software would leave ample space to do more testing at the beginning of the rollout. In retrospect, this was a political decision which should not have been accepted by a security expert. Last but not least, the documentation was found to be complete and consistent, containing a user manual, a setup manual, and configuration instructions for the server as well as for the firewall.

Release. The final security review was done by the security advisor, who found all security requirements to be sufficiently addressed. All potential threats that were ever thought of had been addressed sufficiently, and all bugs that were discovered had been resolved satisfactorily.

Post-SDL. Due to the size of the company and the size of the IT department, the security response plan ended up being quite simple. The about box of the software system provides information about where to send any feedback to the system. Any such information be answered by anyone of the IT department, and will be forwarded to the single developer of that system for further processing.

6. LESSONS LEARNED

We pursued the goal to develop a secure ordering system with a very small development team and to review the suitability of the MS-SDL for that purpose. Using the MS-SDL promises several advantages, i.e., more secure systems but also reduced development costs. It is impossible to make a statement about these issues that are based on hard facts. We are positive that using the MS-SDL even in its downgraded form has made the developed system indeed more secure. We also believe that development costs had not been significantly increased due to this proceeding. We do not see a reduction of costs unless considering ensuing costs that would definitely emerge after the detection of issues in an unsecure system.

Threat modeling had been done with the Visio 2007 tool, which is one of Microsoft's SDL threat modeling tools. At first, use of the tool is quite easy. There are four steps, i.e., "draw diagram", "analyze model", "describe environment", and "generate reports". Step 1 is straightforward. Steps 2 and 3 require quite some input which, of course, influence step 4, the generated reports. A rather small scenario with 12 design elements leads to a report with a total of 53 pages, including threat model, report, bug report, and analysis report. This is an issue, especially when reviewing the threat models. It turned out that the size of these reports does not get bigger to the same extent than the scenarios do. Thus, this is an issue that mostly applies to small software systems and teams. As a consequence, we did not model all the threats with all the details with the tool.

In our case, the final security review turned out to be sort of an acceptance talk between security advisor and developer. This is mostly true, because the security push had been done just before this review and there were no additional development activities rather than error corrections in between. The small size of the development team turned out to be an advantage, because communication between security advisor and developer was frequent during the entire development project.

It has to be mentioned that we did not fully comply with the MS-SDL. This even impossible when a single person is developing software due to the fact that various roles cannot be played or must not be played by the same person. In our case scenario we went to the extreme with only one person in the development team. As mentioned above this fact did have advantages. However, in retrospect we recommend more involved parties for increased security. Security issues have not yet emerged for the developed system. Nevertheless, the more we adhere to the predefined lifecycle, the more we can be sure about the effectiveness of taken security measures. Full adherence is not possible for a single person and should be avoided when security is an issue. Other than that, we strongly recommend to explicitly take security into account and to profit from any secure lifecycle even in small development teams.

7. CONCLUSION

We have used the MS-SDL for the development of a secure web-based ordering system. The size of the development team was small and only consisted of a single developer. The interesting question was whether it was wise to adopt a heavy-weight lifecycle model that is enhanced with security-related issues in our scenario. As it turned out, adaptations have to be made, but using the MS-SDL as a guideline is a definite plus in the pursuit of secure software development.

Various software lifecycle models are available to choose from. They range from classical water-fall models to agile models like extreme programming. These models are hardly ever used exactly as described and without interruption. They get adapted by companies or project teams in order to fulfill their special needs. The same is true for security activities as suggested by the MS-SDL. There is no need to follow everything. But it is highly recommended not to ignore the security issues of the MS-SDL. Nowadays, building secure software is a must. No matter how the software lifecycle looks like that we use to develop software. No matter how big that software is that we have to develop, and no matter how big or small the software development team for that purpose. Security must be on our agenda, and the MS-SDL provides an excellent guideline to accomplish just this.

8. ACKNOWLEDGMENTS

Our thanks go to the Aumayr GmbH, the company that made it possible to develop the system under the given circumstances.

9. REFERENCES

[1] Sommerville I. 2007.*Software Engineering*, 8th edition, Addison-Wesley.

[2] Kent Beck, Cynthia Andres. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional; 2 Edition, 2004.

[3] Alistair Cockburn. Crystal Clear: A Human-Powered Methodology for Small Teams (Agile Software Development), Addison-Wesley Longman, Amsterdam, 2004.

[4] Mike Cohn. Succeeding with Agile: Software Development Using Scrum, Addison-Wesley Signature, Addison-Wesley Longman, Amsterdam, 2009.

[5] Boehm B. and Turner R. 2004. *Balancing Agility and Discipline*, Addison-Wesley.

[6] Swiderski F. and Snyder W., 2004. *Threat Modeling*, Microsoft Press.

[7] Microsoft 2011. *The Microsoft SDL*, www.microsoft.com/security/sdl

[8] OWASP. 2007. OWASP CLASP v1.2. *Comprehensive, Lightweight Application Security Process*.

[9] McGraw G. 2009. *Software Security: Building Security In*, 5th edition, Addison-Wesley.

[10] Microsoft 2010. *Security Development LifeCycle V. 5.0*, Microsoft Press.

[11] Howard M. Lipner S. 2006. *The Security Development Lifecycle*, Microsoft Press.

[12] Curphey M., Scambray J., Olson E. 2003. *Improving Web Application Security: Threats and Countermeasures*, Microsoft Press.

[13] Howard M., LeBlanc D. 2003. *Writing Secure Code*, Microsoft Press.

[14] Microsoft. *Improving Web Application Security: Threats and Countermeasures*, http://msdn.microsoft.com/en-us/library/ff649874.aspx

[15] Hernan, S., Lambert, S., Ostwald, T., and Shostack, A. 2006. Uncover Security Design Flaws Using The STRIDE Approach. *MSDN Magazine*, Nov. 2006, http://msdn.microsoft.com/en-us/magazine/cc163519.aspx