SECURITY STANDARDS IN SOFTWARE DEVELOPMENT

L. Petkova, G.

University of Library Studies and Information Technologies (UNIBIT) Bulgaria

Abstract

Those interested in software security are oriented in direction of composing security standards of information and every activity of the product. Those standards help the developers to produce and the contractor to deliver a software corresponding the highest levels of privacy and data security.

The software development is based on the requirements of the guarantor determined the goal and tasks to achieve it. This is the begging of the structuring and developing the software. However, the developer's work does not end with deploying the software, but to deliver software which may resist all kinds of attacks or incorrect user activities during his software reaction.

That article can be useful for developers interested in improving the security quality of a software product in the process of developing by using some security standards.

Keywords: standards, security, software, development, integration, cyber security, confidentiality, data, protection.

1 INTRODUCTION

The goal of the article is to present several standards and different methodologies of software developing by integrating security in every phase of the life cycle.

Firstly, we will basically describe two methodologies for developing software by implementing security in every phase of the life cycle of software development: Analyses, Structuring, Developing, Testing and Deploying! After that we will present several security standards in two categories: Security Standards in SDLC and Standards in Data Security! Finally, we will give basic information for two frameworks of software security, presented by the governments of Bulgaria and of USA, based on the data defense in Internet!

The success of a software product depends both on the level of correspondence of the primary requirements, provided by the guarantor, and the sustainability of either intentional or unintentional attack! By offering sets of tools shared in the developers' worldwide the software development is more organized and clearer in terms of protecting it.

2 METHODOLOGIES

Security tends to be associated with a coding project in three major areas. The first is during the QA reviews, where bugs are found, logged and prioritized. Although this is a somewhat logical approach, its success relies on a good working relationship between the development and QA teams to develop functional reviews and proper code analysis, which usually involves more than one technique. [4]

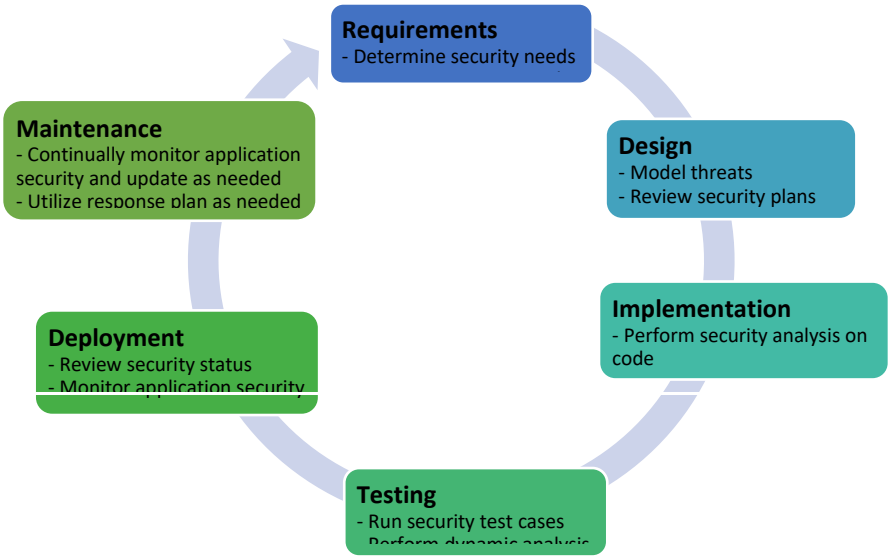
The second major integration point for security tends to be the "security toll gate," where security team members are often one part of a project review committee or planning board. Although the toll gate scenario gives security professionals some insight into the project itself, it is at a high level that may not allow them to be effective. In fact, this approach ultimately relies on security being prioritized in a top-down manner from senior leadership, which is rare. [4][15]

The third and most comprehensive method integrates security and risk management into the development cycle, either with involvement of security team members or by developing a core security focus on the part of developers. [4][15]

Several well-known Software Development Life Cycles (SDLCs) integrate security in different ways. The first is the traditional waterfall SDLC, which has been used for many years. Although many versions of this model exist, the SDLC generally starts with a requirements specification phase followed by design, implementation, testing, deployment and maintenance phases. [4][15]

Security can be integrated into any (and ideally all) of these phases. In most organizations that use a variant of the waterfall model, security is included with the toll gate style, often at the end of each phase before moving to the next one. It is, however, critically important to ensure that security is prioritized during the requirements specification phase and carried out at every phase, particularly in organizations where developers and QA teams are responsible for policing themselves (see Figure 1). [4][15]

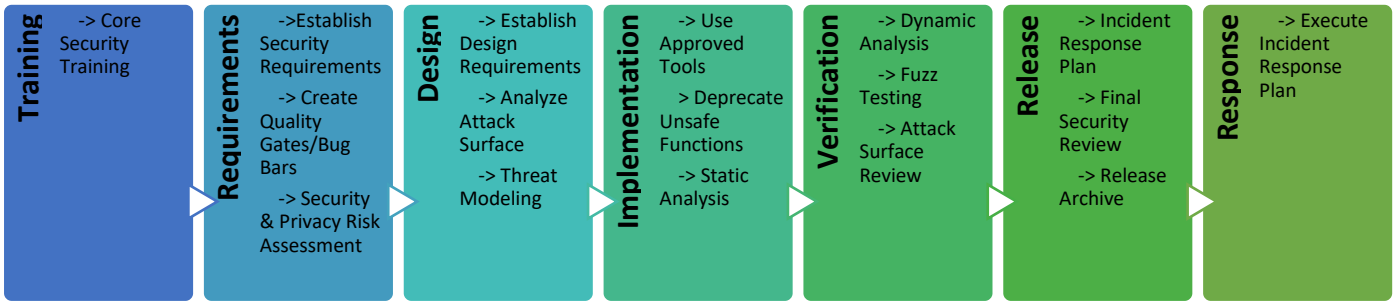
If security drives the software development life cycle process, the responsible parties must work with developers to determine their needs and provide input during every step. This process enables the applications to be built securely while helping development teams to maintain a reasonable schedule. [4][15]



2.1 Security Development Lifecycle Microsoft

Security Development Lifecycle Microsoft is a good example of a company that was ultimately forced to adapt its SDLC due to the large number of vulnerabilities found in its flagship Windows operating system between 1999 and 2003. Microsoft has steadily improved security ever since—and continues to do so using its own development model. [5][15]

Microsoft uses a popular adaptation of the waterfall SDLC that was adapted for tighter security integration and was dubbed the Security Development Lifecycle (SDL). This model, illustrated in Figure 2, incorporates security training for developers before the requirements specification phase, as well as separate verification (prerelease) and response (post release) stages.7 For large development teams with extensive resources, particularly those in companies that sell software to the masses, this



model may make more sense, because security is vitally important throughout the entire coding and QA project, every time. [5][15]

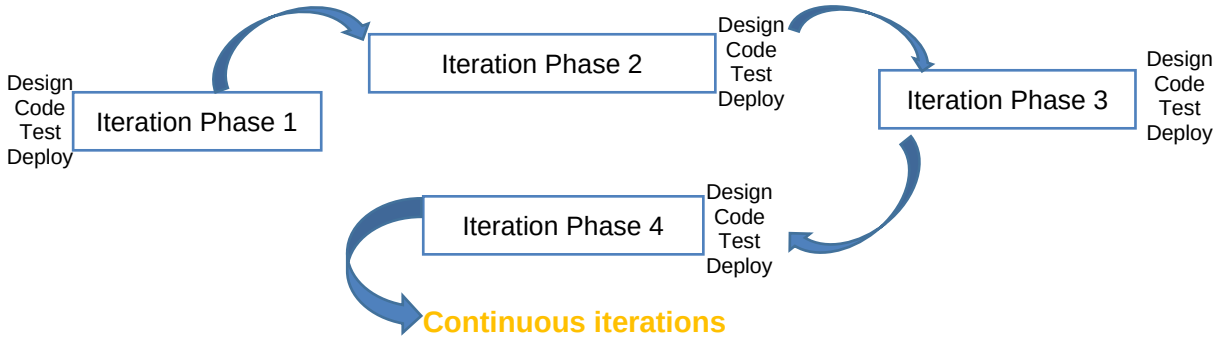
Key points in Microsoft Security Development Lifecycle (MSDLC) are: [3][15]

- MSDLC is Microsoft's security assurance process for software development that builds security into every phase of software development and provides defense-in-depth guidance and protection.

- The SDL is a hands-on set of procedures involving testers, developers, program managers, and architects working in concert with product security teams across the company. Its security innovations are integrated into Microsoft Office, the Windows operating system, Microsoft SQL Server, and many other Microsoft products and services.
- The SDL is continuously evolving and improving. It is updated to take advantage of newly developed defensive techniques in security science and in anticipation of emerging threats.
- Microsoft shares the SDL with the software industry. The SDL has been adopted (sometimes in a modified form) by a variety of software and hardware vendors, government agencies, and software development organizations.

2.2 Agile Development

Finally, a third widely-used development cycle, known as Agile, focuses on personal interactions among development team members and rapid response to change. In the Agile SDLC, shown in Figure 3, pushing code quickly and responding to new feature and code change requests quickly is a priority, as is code and object reuse, and this can come into direct conflict with security principles. In an Agile organization, the most reasonable method for integrating security is either constant involvement from security team members or a dedication to security with well-trained coders on every team. [5][15]



2.3 Scalable and Agile Lifecycle Security for Applications

More organizations are working toward improving the synchronization between security and development teams, as well as changing the way developers work overall. A paper published in 2007 by SANS introduced a new methodology called Scalable and Agile Lifecycle Security for Applications (SALSA). Although not a true development life cycle in itself, SALSA allows web application developers, in particular, to start strategically incorporating security into the existing life cycles and adapting them for improved code review and analysis. Several key tenets of SALSA include the following policies for bringing security and developers together during the SDL: [5][15]

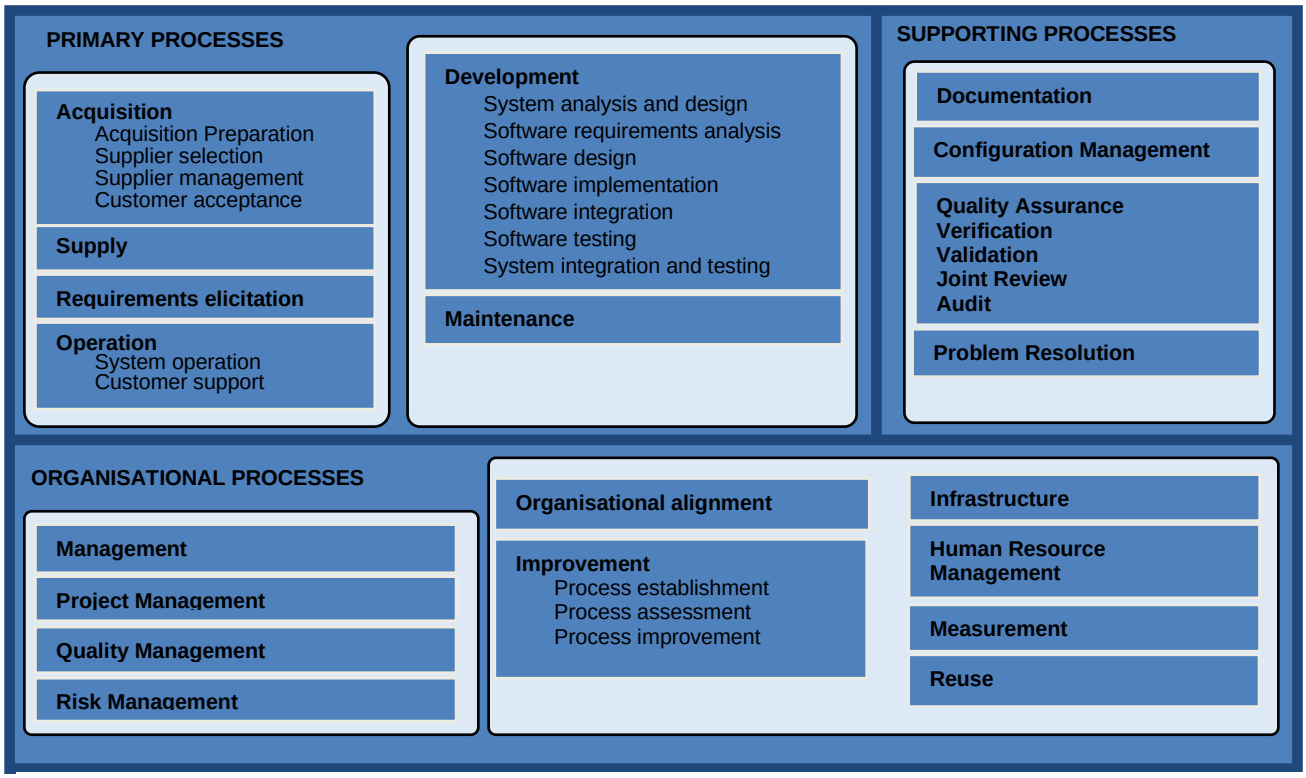
- **Educate developers on how to analyze the enterprise attack surface or application, as well as the associated potential threats**. Although this suggestion sounds like more of a job for the security or risk-analysis team, all developers should understand the exposure points of their applications (user inputs, web-facing code, exposed function calls and so on) and take steps to design more secure code wherever possible.
- **Integrate security best practices into application life cycle and development methodologies**. This approach was largely pioneered by the Microsoft SDL mentioned previously, but SALSA also incorporates attack surface analysis and reduction.
- **Integrate security into the automated build process**. This suggestion is excellent practical advice, especially for Agile development teams, because automated builds are a core part of the rapid change and-response element of the Agile Manifesto. The automated build process should include some general consideration of code complexity (metrics-based or otherwise), as well as automated code and unit testing using the tools discussed previously.
- **Procure security training for developers and some development training for security professionals, depending on their needs**. This training will be more of an ongoing activity that changes from year-to-year.

- **Include automated vulnerability assessments and penetration testing whenever possible**. Such assessment helps significantly with analysis of the application attack surface and provides new insights into potential vulnerabilities that might have been missed in earlier code reviews and testing.
- **Adopt more transparency**! Make it simpler for customers (both internal and external) to submit bugs, ensure all stakeholders have a clear way of getting to project information (phases, tasks, people) and provide feedback to users on progress made in development, particularly with regard to security

3 OVERVIEW OF THE SECURITY STANDARDS IN SOFTWARE DEVELOPMENT

3.1 Standards for Software Life Cycle Process

ISO/IEEE 12207 is an international software engineering standard that defines the software engineering process, activity, and tasks that are associated with a **software life cycle process** from conception through retirement**.** It aims to be 'the' standard that defines all the tasks required for developing and maintaining software. [6]



The standard is based on two basic principles: modularity and responsibility. Modularity means processes with minimum coupling and maximum cohesion. Responsibility means to establish a responsibility for each process, facilitating the application of the standard in projects where many people can be legally involved. [6]

The life cycle processes are grouped into three broad classes: primary; supporting; and organizational. Primary processes are the prime movers in the life cycle; they are acquisition, supply, development, operation, and maintenance. Supporting processes are documentation, configuration management, quality assurance, joint review, audit, verification, validation, and problem resolution. A supporting process supports another process in performing a specialized function. Organizational processes are management, infrastructure, improvement, and training. An organization may employ an organizational process to establish, control, and improve a life cycle process. [6]

Each process is further designed in terms of its own constituent activities, each of which is further designed in terms of its constituent tasks. An activity under a process is a set of cohesive tasks. [6]

A task is a set of elementary or atomic actions. A task consumes inputs (data, information, control) and produces outputs (data, information, control). It is expressed in the form of self-declaration, requirement, recommendation, or permissible action. For this purpose, the standard carefully employs certain auxiliary verbs (will, shall, should, and may) to differentiate between the distinct forms of a task. "Will" is used to express a self-declaration of purpose or intent by one party, "shall" to express a binding provision between two or more parties, "should" to express a recommendation among other possibilities, and "may" to indicate a course of action permissible within the limits of the standard. "Must," which denotes a mandatory action, is never used in the standard. [5]

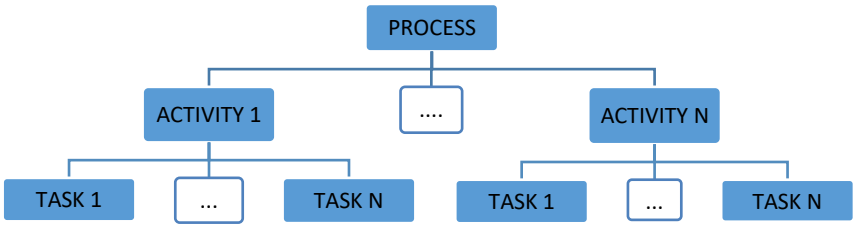


Figure 5. Process composition[5]

ISO/IEEE 15288:2015, "This International Standard establishes a common framework of process descriptions for describing the life cycle of systems created by humans. It defines a set of processes and associated terminology from an engineering viewpoint. These processes can be applied at any level in the hierarchy of a system's structure. Selected sets of these processes can be applied throughout the life cycle for managing and performing the stages of a system's life cycle. This is accomplished through the involvement of all stakeholders, with the goal of achieving customer satisfaction. [2]

This International Standard also provides processes that support the definition, control and improvement of the system life cycle processes used within an organization or a project. Organizations and projects can use these processes when acquiring and supplying systems. [2]

This International Standard concerns those systems that are man-made and may be configured with one or more of the following system elements: hardware, software, data, humans, processes (e.g., processes for providing service to users), procedures (e.g., operator instructions), facilities, materials and naturally occurring entities. [2]

When a system element is software, the software life cycle processes in ISO/IEC/IEEE 12207:2015 may be used to implement that system element. The two standards are harmonized for concurrent use on a single project or in a single organization. [2]

Despite the advantages of this standard there is some limitation of its use. This International Standard does not prescribe a specific system life cycle model, development methodology, method, model or technique. The users of this International Standard are responsible for selecting a life cycle model for the project and mapping the processes, activities, and tasks in this International Standard into that model. The parties are also responsible for selecting and applying appropriate methodologies, methods, models and techniques suitable for the project. [2]

ISO/IEC 15504, also known as SPICE (Software Process Improvement and Capability Determination), is a framework for the assessment of processes. [4]

SPICE is a major international initiative to support the development of an International Standard for Software Process Assessment. The project has three principal goals:

- To develop a working draft for a standard for software process assessment
- To conduct industry trials of the emerging standard
- To promote the technology transfer of software process assessment into the software industry world-wide

The first goal was achieved on June 1995 when the version 1 (draft standard) was released. By the normal process of development of international standards, the SPICE documents have been published as ISO/IEC TR 15504:1998 - Software Process Assessment. WG10 is continuing the work to the goal, full international standard. [4]

SPICE provides a set of documents, which are used as a framework for the assessment of software process. Organizations can use these documents in various phases of production, for example in

planning, managing, monitoring, controlling and improving acquisition, supply, development, operation, evolution and support of software. [4]

Basically, software process assessment examines the selected processes whether they are effective in achieving their goals, which is done by determining the capability of the selected processes. This structured approach for software process assessment helps an organization to improve its processes or to determine its capability for certain requirement, or to determine suppliers' capability for certain requirement. [4]

Process assessment provides information of the capability of the selected processes. Analysis results, from business point of view, identify strengths, weakness and risks inherent in the processes. By this, analysers can determine whether the processes are effective, and to identify significant causes of poor quality, or over runs in time or cost. After recognizing these kinds of issues, managers can prioritise improvements to processes. [4]

Process capability determination analyses the proposed capability of selected processes against a target process capability profile. By this, it tries to find out the risks involved in a project, if the project is run with the analysed processes. [4]

The document suite of SPICE contains nine different documents, which can be used in process assessment. These documents will be shortly described in the following subchapters. [4]

SPICE documents from 1 to 6 mainly concentrate in addressing various aspects related to process assessment. Other documents, 7 and 8, address the use of process assessment for process improvement or for process capability determination. Document 9 acts as a vocabulary. [4]

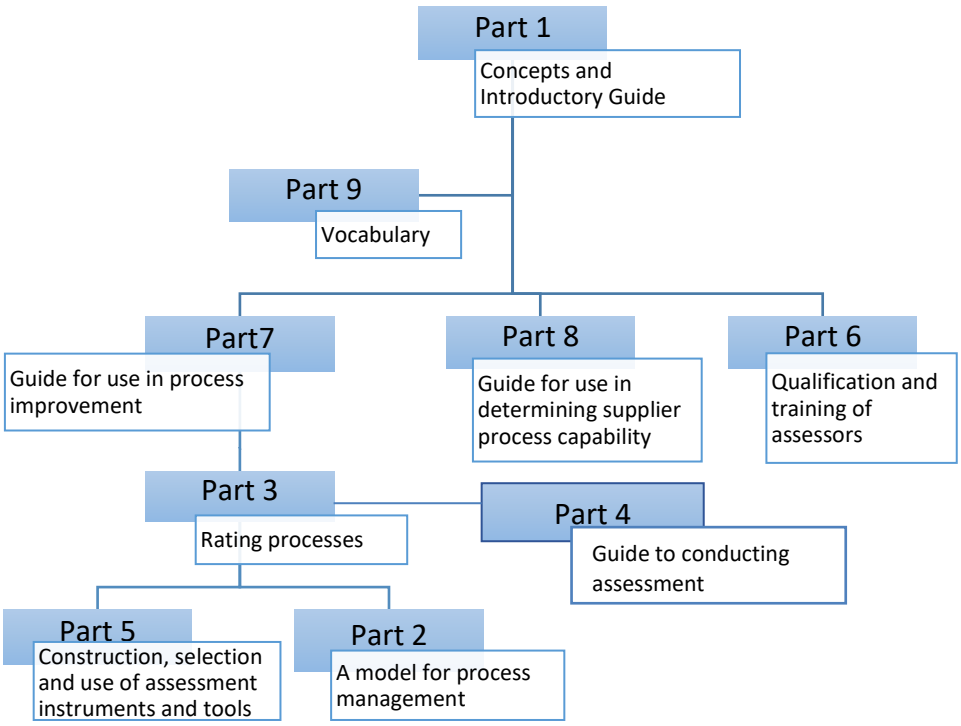


Figure 6. SPICE documents[4]

3.2 Standards for Information Security

ISO/IEC 27000

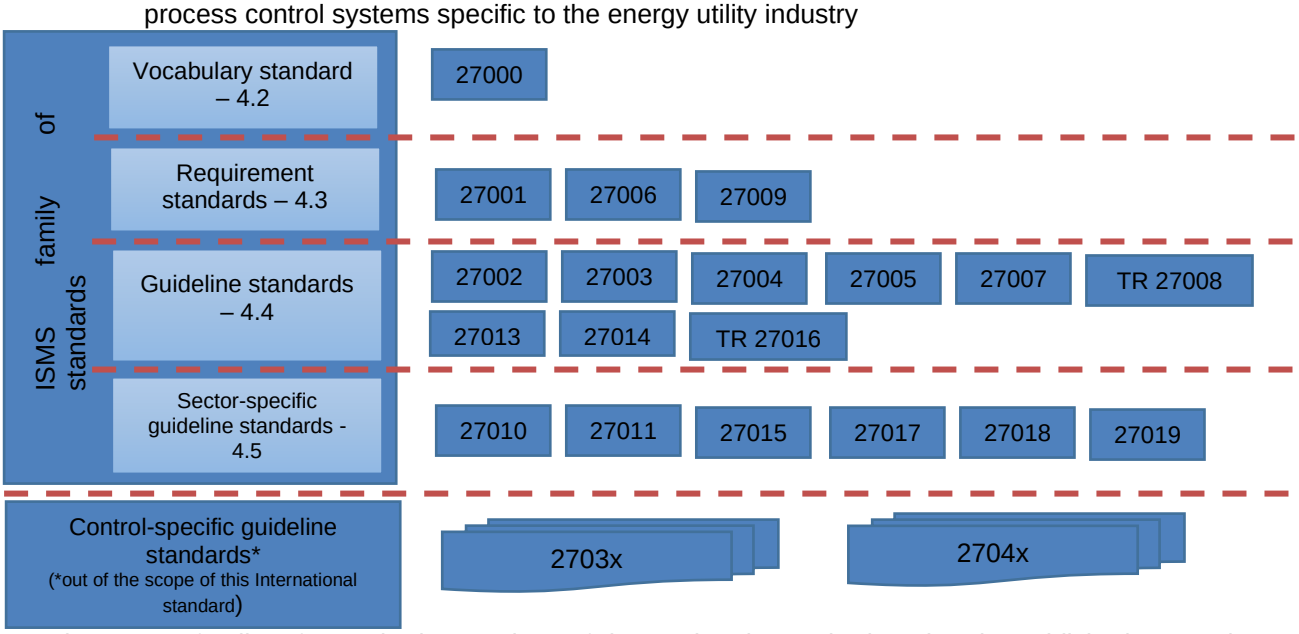
International Standards for management systems provide a model to follow in setting up and operating a management system. This model incorporates the features on which experts in the field have reached a consensus as being the international state of the art. ISO/IEC JTC 1/SC 27 maintains an expert committee dedicated to the development of international management systems standards for information security, otherwise known as the Information Security Management System (ISMS) family of standards. [1]

Using the ISMS family of standards, organizations can develop and implement a framework for managing the security of their information assets including financial information, intellectual property,

and employee details, or information entrusted to them by customers or third parties. These standards can also be used to prepare for an independent assessment of their ISMS applied to the protection of information. [1]

The ISMS family of standards is intended to assist organizations of all types and sizes to implement and operate ISMS and consists of the following International Standards, under the general title Information technology — Security techniques (given below in numerical order): [1]

- **ISO/IEC 27000**, Information security management systems — Overview and vocabulary
- **ISO/IEC 27001**, Information security management systems — Requirements
- **ISO/IEC 27002**, Code of practice for information security controls
- **ISO/IEC 27003**, Information security management system implementation guidance
- **ISO/IEC 27004**, Information security management — Measurement
- **ISO/IEC 27005**, Information security risk management
- **ISO/IEC 27006**, Requirements for bodies providing audit and certification of information security management systems
- **ISO/IEC 27007**, Guidelines for information security management systems auditing
- **ISO/IEC TR 27008**, Guidelines for auditors on information security controls
- **ISO/IEC 27009**, Sector-specific application of ISO/IEC 27001 — Requirements
- **ISO/IEC 27010**, Information security management for inter-sector and inter-organizational communications
- **ISO/IEC 27011**, Information security management guidelines for telecommunications organizations based on ISO/IEC 27002
- **ISO/IEC 27013**, Guidance on the integrated implementation of ISO/IEC 27001 and ISO/IEC 20000-1
- **ISO/IEC 27014**, Governance of information security
- **ISO/IEC TR 27015**, Information security management guidelines for financial services
- **ISO/IEC TR 27016**, Information security management — Organizational economics
- **ISO/IEC 27017**, Code of practice for information security controls based on ISO/IEC 27002 for cloud services
- **ISO/IEC 27018**, Code of practice for protection of personally identifiable information (PII) in public clouds acting as PII processors
- **ISO/IEC 27019**, Information security management guidelines based on ISO/IEC 27002 for process control systems specific to the energy utility industry



The ISMS family of standards consists of inter-related standards, already published or under development, and contains several significant structural components. These components are requirements (ISO/IEC 27006) for those certifying conformity with ISO/IEC 27001, and additional requirement framework for sector-specific implementations of the ISMS (ISO/IEC 27009). Other standards provide guidance for various aspects of an ISMS implementation, addressing a generic process as well as sector-specific guidance. [1]

Relationships between the ISMS family of standards are illustrated in Figure 7.

3.3 Code Security Standards

Secure coding standards define rules and recommendations to guide the development of secure software systems. Establishing secure coding standards provides a basis for secure system development as well as a common set of criteria that can be used to measure and evaluate software development efforts and software development tools and processes.[11][13]

The CERT Division has been extremely successful in the development of secure coding standards, which have been adopted at corporate levels by companies such as Cisco and Oracle, and the development of the Source Code Analysis Laboratory (SCALe), which supports conformance testing of systems against these coding standards. The success of the secure coding standards and SCALe contributed to the impetus for including software assurance requirements in the National Defense Authorization Act (NDAA) for Fiscal Year 2013. [10][13]

There are some practices related to the security of the programming code. Here we present top 10 of the secure coding practices part of the measurements against the software vulnerabilities:[13][14]

1. **Validate input.** Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files.[9] [12][13][14]
2. **Heed compiler warnings.** Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.
3. **Architect and design for security policies**. Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. [9][13][14]
4. **Keep it simple.** Keep the design as simple and small as possible. [7][8] Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. [9][13][14]
5. **Default deny.** Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted. [7][8][13][14]
6. **Adhere to the principle of least privilege.** Every process should execute with the the least set of privileges necessary to complete the job. Any elevated permission should be held for a minimum time. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges. [7][8] [13][14]
7. **Sanitize data sent to other systems.** Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components using SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. [9][13][14]
8. **Practice defense in depth.** Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment. [9] [13][14]
9. **Use effective quality assurance techniques.** Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Penetration testing, fuzz testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions. [9] [13][14]
10. **Adopt a secure coding standard.** Develop and/or apply a secure coding standard for your target development language and platform. [9][13][14]

Everyday software defects cause the majority of software vulnerabilities. [11][13]

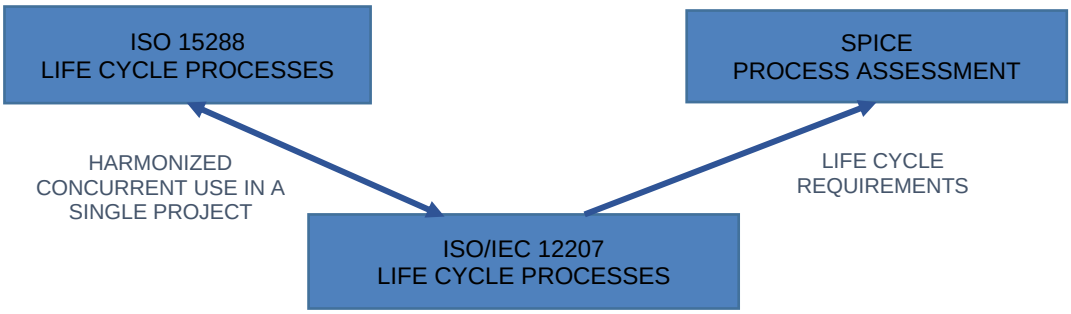
Software developers are not always properly trained Software developers are not always properly trained and equipped to program securely. [11]

The result is numerous delivered defects, some of which can lead to vulnerabilities. [11][13]

Understanding the sources of vulnerabilities and learning to program securely is imperative to protecting the Internet and ourselves from attack. [11][13][14]

4 RESULTS

Relationships among ISO 15288, SPICE and ISO/IEC 12207. At the process level, the ISO, IEC and JTC1 have been active in the standardization of quality assurance, process assessment, and life cycle processes. The former has been accomplished at the system level; the latter two are being pursued at the software level. ... depicts the relationships among quality assurance, process assessment, and life cycle processes standardizations. We modified the diagram to work with the three life cycle standards described in the current article. At the top left corner, ISO 15288 represents the processes employed throughout the life cycle of a software product. At the top right corner, SPICE represents process assessment as applied in organizations. At the bottom center, ISO/IEC 12207 represents the processes employed throughout the life cycle of a software product. ISO 15288, as the arrows from it show, provides the same logic of ISO/IEC 12207. They can be concurrent used in a single project. ISO/IEC 12207 provides (see the arrow from it to SPICE) the baselines of life cycle processes to the SPICE assessment process. It is expected that as these standards evolve in the future, their mutual consistencies would improve.[6]



5 CONCLUSIONS

The use of standards is voluntary! It does not in itself impose any obligation upon anyone to follow it. Yet, it may be imposed by an organization through internal policy directive or by individual parties through contractual agreements. The standard is designed for use by one or more parties as the basis of an agreement or in a self-imposed way.

ACKNOWLEDGEMENTS

REFERENCES

[1] International Standards ISO/IEC 27000, Information technology — Security techniques — Information security management systems — Overview and vocabulary. (2015-2016). NIST.

[2] International Standards ISO/IEC/ IEEE 15288, Systems and software engineering — System life cycle processes. (2015). NIST.

[3] Microsoft Security Development Lifecycle. (2013, February).

[4] Pyhjrvi, M. (2004, November). SPICE International Standard for Software Process Assessment.

[5] Shackleford, D. (2011). Integrating Security into Development, No Pain Required. *SANS Institute InfoSec Reading Room*.

[6] Singh, R. (n.d.). INTERNATIONAL STANDARD ISO/IEC 12207 SOFTWARE LIFE CYCLE PROCESSES. USA, DC: Federal Aviation Administration Washington.

[7] Saltzer, J. H. "Protection and the Control of Information Sharing in Multics." Communications of the ACM 17, 7 (July 1974): 388-402.
[8] Saltzer, J. H. & Schroeder, M. D. "The Protection of Information in Computer Systems." Proceedings of the IEEE 63, 9 (September 1975), 1278-1308.
[9] Seacord, R. Secure Coding in C and C++. Upper Saddle River, NJ: Addison-Wesley, 2006 (ISBN 0321335724).
[10] CERT/CC. Secure Coding website. http://www.cert.org/secure-coding/
[11] Seacord, Robert, Rafail, Jason. (2008, December) Secure Coding Standards
[12] OWASP Secure Coding Practices Quick Reference Guide (2010, August)
[13] ICT Security Trends, Willian Dimitrov, Sofia, 2017, Avangard, ISBN 978-619-160-766-2
[14] Software testing, Willian Dimitrov, Sofia, 2017, Avangard, ISBN 978-619-160-765-5
[15] ICT Security Model, Willian Dimitrov, Sofia, 2018, Avangard, ISBN 978-619-160-950-5