Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 51 (2009) 1152–1171
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

On the secure software development process: CLASP, SDL and Touchpoints compared

Bart De Win *, Riccardo Scandariato, Koen Buyens, Johan Grégoire, Wouter Joosen

Department of Computer Science, DistriNet, K.U. Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium

Available online 15 February 2008

Abstract

Development processes for software construction are common knowledge and mainstream practice in most development organizations. Unfortunately, these processes offer little support in order to meet security requirements. Over the years, research efforts have been invested in specific methodologies and techniques for secure software engineering, yet dedicated processes have been proposed only recently.

In this paper, three high-profile processes for the development of secure software, namely OWASP's CLASP, Microsoft's SDL and McGraw's Touchpoints, are evaluated and compared in detail. The paper identifies the commonalities, discusses the specificity of each approach, and proposes suggestions for improvement.

Keywords: Secure software; Software process; SDL; CLASP; Touchpoints

1. Introduction

The construction of secure software is still largely a matter of guidelines, best practices and undocumented expert knowledge. Current practices provide guidance for particular areas such as threat modeling, risk management, or secure coding. It is crucial for these to be combined into an integrated and more comprehensive construction method. Several advances have recently been made in the definition of processes for secure software development. However, there has been no objective comparison of these methodologies so far. Therefore, it is difficult for managers and developers to understand their strengths and weaknesses and, hence, it is hard to make an 'informed' decision about which one is more appropriate for the job.

The goal of this paper is to evaluate and compare processes for secure software development. The focus is put on three forefront representatives, namely Microsoft's *Security Development Life cycle* (SDL) [1], OWASP's *Comprehensive, Lightweight Application Security Process* (CLASP) [2] and McGraw' *Touchpoints* [3], as they are recognized as the major players in the field. Their leading role is, among others, due to a number of characteristics that are of particular interest in the context of this study. As far as completeness is concerned, they all provide an extensive set of activities covering a broad spectrum of the development life-cycle. The processes have also undergone extensive validation. For instance, Microsoft is using SDL internally (e.g., for the Vista project). CLASP was contributed to and reviewed by several leading security companies of the OWASP consortium. The Touchpoints work has been validated over time as it has been based on experience gained from several industrial projects. Furthermore, the selected processes represent a healthy mix for comparison. SDL is considered to be more heavyweight and rigorous, making it more suitable for large organizations. CLASP, on the other hand, is lightweight and more affordable for small organizations with less strict security demands. Touchpoints is based on industrial experience

* Corresponding author. Tel.: +32 16 327855; fax: +32 16 327996.
E-mail address: first.last@cs.kuleuven.be (B. De Win).

gathered over several years which was subsequently distilled into an approach, making sure that the proposed activities are both executable and feasible. Finally, the selection of the processes was also influenced by the availability of thorough documentation.

This paper compares CLASP, SDL and Touchpoints, mainly from a theoretical perspective. A first contribution is the *activity-matrix*, which lists and relates all activities of the different processes in a comprehensive model. The matrix facilitates the identification of similarities between the processes and it introduces grouping of activities into regular development phases. A second contribution is the actual comparison of the processes from different perspectives: (i) generic characteristics of the processes are discussed in order to outline the philosophy underlying a particular method; (ii) activity-wise, the commonalities and the differences between the methods are identified and discussed in order to articulate the strong and weak points of these methods and (iii) process-wise, a number of improvements are outlined that the processes could benefit from, in terms of both coverage and quality. In order to complement the theoretical discussion, the results of applying the three processes on a small-scale case study are presented and they confirm the findings of the theoretical study.

The structure of this paper is as follows. Section 2 describes the methodological approach that was adopted for the comparison. Section 3 contains a bird's eye presentation of the processes, with a particular focus on their characterizing philosophy. Section 4 zooms into both the commonalities and the differences of the processes, on a phase-by-phase basis. Section 5 summarizes the results of the application of the different processes on a small-scale case study. Section 6 presents a number of challenges for the presented processes and highlights the room for improvement. Section 7 discusses related work. Finally, Section 8 presents the conclusions as well as directions for future work.

2. Methodology

This section briefly sketches the methodology that was used for the comparison of the selected processes. To begin, the team agreed on the sources to be used, since possible differences and inconsistencies may exist in different versions. For CLASP it was decided to use the documentation of version 1.2 available at the OWASP website.[1] For SDL the book by Howard and Lipner [1] was used and for Touchpoints the book by McGraw [3].

Based on the available sources, three by-the-book initial lists of activities were articulated. For some activities this required some interpretation of text to elicit implicit activities. Furthermore, some clean-up was necessary on these lists. This meant optimizing the hierarchical structure of activities and similar transformations. Once the lists of activities were agreed upon, they were merged by linking activities that are conceptually similar. This was not always straightforward, as the granularity and order of activities is not always identical and as particular activities (such as threat modeling) can have many different interpretations. Of all steps taken, this was probably the most challenging one, and the goal was to be as precise as possible.

To increase the structure and the organization of activities, the activities were subsequently organized into the different phases of a typical software development process ('Education and Awareness', 'Project Inception', 'Analysis and Requirements', 'Architectural Design', 'Detailed Design, 'Implementation', 'Testing', 'Release and Deployment' and 'Support'). Since SDL activities are already organized into stages, the activities were mapped onto the above-mentioned phases using extra information provided by MSDN [4]. As far as CLASP is concerned, activities have been assigned to phases by looking at the role that is responsible for the activity itself. Finally, Touchpoints provides a mapping of the touch points to typical software development phases, which was used accordingly.

The result of this analysis was structured as a matrix, which contains a section for each of the aforementioned phases, with each row representing a certain activity and each column representing one of the approaches. Fig. 1 shows a fragment of the matrix, focusing on the 'Detailed Design' phase. The complete matrix has been included in Appendix A. The matrix has been the foundation for the further analysis and comparison of the candidate processes, in particular (i) to identify the intersection and the differences between them and (ii) to apply them to the case study.

3. Background and general characteristics

In this section, the processes are further introduced and a number of characteristics are discussed to give a flavor of their overall philosophy.

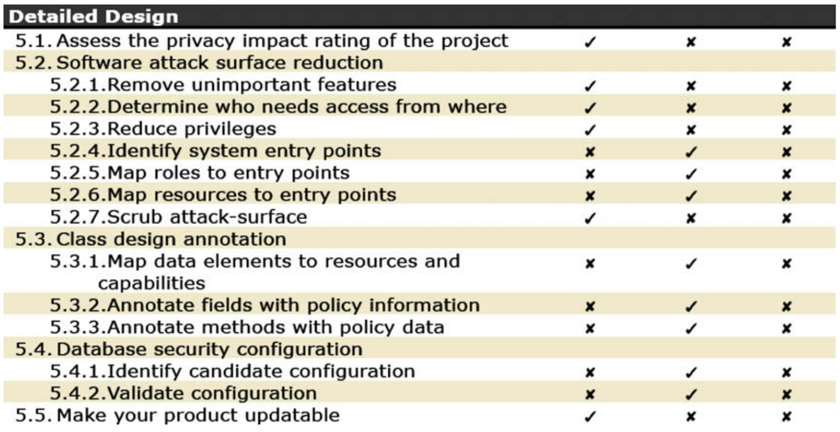
3.1. CLASP

Originally defined by Secure Software and later donated to OWASP, CLASP is a lightweight process for building secure software [2]. It includes a set of 24 top-level activities, which can be tailored to the development process in use. Key characteristics include:

Security at the center stage: The primary goal of CLASP is to support the construction of software in which security takes a central role. Furthermore, the activities of CLASP are defined and conceived primarily from a security-theoretical perspective and, hence, the coverage of the set of activities is fairly broad.

Limited structure: CLASP is defined as a set of independent activities that have to be integrated in the development process and its operating environment. The choice of the activities to be executed and the order of execution is left

[1] CLASP 2 has been announced for some time now, but no new material is available from the website yet.



Detailed Design			
5.1. Assess the privacy impact rating of the project	✓	✗	✗
5.2. Software attack surface reduction			
5.2.1. Remove unimportant features	✓	✗	✗
5.2.2. Determine who needs access from where	✓	✗	✗
5.2.3. Reduce privileges	✓	✗	✗
5.2.4. Identify system entry points	✗	✓	✗
5.2.5. Map roles to entry points	✗	✓	✗
5.2.6. Map resources to entry points	✗	✓	✗
5.2.7. Scrub attack-surface	✓	✗	✗
5.3. Class design annotation			
5.3.1. Map data elements to resources and capabilities	✗	✓	✗
5.3.2. Annotate fields with policy information	✗	✓	✗
5.3.3. Annotate methods with policy data	✗	✓	✗
5.4. Database security configuration			
5.4.1. Identify candidate configuration	✗	✓	✗
5.4.2. Validate configuration	✗	✓	✗
5.5. Make your product updatable	✓	✗	✗

Fig. 1. Excerpt of the matrix (the complete matrix is included in Appendix A).

open for the sake of flexibility. Moreover, the execution frequency of activities is specified per individual activity and, hence, the coordination and synchronization of activities is not straightforward. Two road maps ('legacy' and 'greenfield') have been defined to give some guidance on how to combine the activities into a coherent and ordered set.

Role-based: CLASP defines the roles that can have an impact on the security posture of the software product and assigns activities to these roles. Roles are responsible for the finalization and the quality of the results of an activity. As such, roles are used as an additional perspective to structure the set of activities.

Rich in resources: CLASP provides an extensive set of security resources that facilitate and support the implementation of the activities. For instance, one of these resources is a list of 104 known security vulnerabilities in application source code (e.g., to be used as a checklist during code reviews).

3.2. SDL

As a result of its commitment to trustworthy computing proclaimed in 2002, Microsoft defined the SDL to address the security issues they frequently faced in their products. SDL comprises a set of activities, which complement Microsoft's development process and which are particularly aimed at addressing security issues. SDL can be characterized as follows:

Security as a supporting quality: The primary goal of SDL is to increase the quality of functionality-driven software by improving its security posture. Security activities are most often related to functionality-based construction activities. For instance, threat modeling starts from architectural dependencies with external systems, while an architecture could in fact reduce such threats in the first place. SDL is designed as an add-on to the software construction process.

Well-defined process: The SDL process is well organized and related activities are grouped in stages. Although these stages are security specific, it is straightforward to map them to standard software development phases. Furthermore, several activities have a continuous characteristic in the SDL process, including threat modeling and education. As such, the SDL process incorporates support for revising and improving intermediate results.

Good guidance: SDL does a good job at specifying the method that must be used to execute activities, which, on average, are concrete and often somewhat pragmatic. For instance, attack surface reduction is guided by a flow chart and threat modeling is described as a set of sub-processes. As a result, the execution of an activity is quite achievable, even for less experienced people.

Management perspective: SDL takes a management perspective for the elicitation and description of many activities. This is nice, given the inherent complexity of security, and it shows that security as a quality has to be managed in order to be realized in practice.

3.3. Touchpoints

Touchpoints provides a set of best practices that have been distilled over the years out of the extensive industrial experience of its proposer. Most of the best practices, named activities from here on, are grouped together in seven so-called touch points. Touchpoints can be characterized as follows:

Risk Management: Touchpoints acknowledges the importance of risk management when it comes to software security. It tries to bridge the gap by elaborating a Risk Management Framework (RMF) that supports the Touchpoints activities.

Black vs. White: The touch points provide a mix of black-hat and white-hat activities, both of which are necessary to come to effective results. Black-hat activities are about attacks, exploits and breaking software (e.g., penetration testing). White-hat activities are more constructive in nature and cover design, controls and functionality (e.g., code review).

Flexibility: The touch points can be tailored to the software development process already in use. To facilitate this, the documentation provides a prioritization of the different touch points. This allows companies to gradually introduce the touch points, starting from the most important ones.

Examples: Touchpoints is rich on examples. For instance, when describing abuse cases, there is an example giving the reader a good feel about what they might look like in a particular situation.

Resources: To further aid the execution of activities, Touchpoints provides links to resources and also explains how to use them. To this aim, a part of the book is dedicated to security knowledge (which the resources are part of). For instance, attack patterns are provided in order to be used in the elicitation of abuse cases.

4. Phase-by-phase comparison

In this section, a comparison of the three processes on the level of activities is presented. The activities have been classified according to software development phases, as discussed in Section 2. Some phases have been merged in this section, however, for reasons of readability. For each phase the processes are discussed, pointing out commonalities and differences. The goal of the comparison in this section is to discuss the important differences, not to be exhaustive. The reader is referred to the complete matrix in Appendix A that may serve as a source for an exhaustive comparison.

4.1. Education and awareness

SDL: Education is a fundamental part of SDL. Every team member of the project should be given *baseline education* in software security (Activity 1.1) in order to increase (i) the awareness of the importance of the problem and its broad scope and (ii) the knowledge of security engineering basics, including core concepts, types of security breaches, possible solutions, and so on.

More focused and targeted *advanced education* (Activity 1.3) is scheduled as well for particular groups of engineers. As security is a rapidly evolving field, with new threats emerging frequently, such courses are scheduled annually. Some examples of courses in this area include fuzz testing, threat modeling and reviewing code. An infrastructure for *tracking the attendance* (Activity 1.3.2) and for *measuring* the effectiveness of courses (Activity 1.3.3) is suggested.

CLASP: Similar to SDL, CLASP emphasizes the importance of education (base-level as well as advanced). However, CLASP has a somewhat different focus in that it addresses all project roles (e.g., including the project manager). Moreover, accountability is a key driver to organize courses, since engineers can only be held responsible for security problems when they are sufficiently trained. CLASP also improves security awareness by proactively *sharing all security artifacts* within the development team (Activity 1.4).

Touchpoints: Since education is not one of the seven core touch points, little attention is given to the topic. It is recognized that people should be sufficiently trained, especially about the particularities of the development environment as software engineers are considered to be more in line with the Touchpoints philosophy. The information security people are the main target for training. Besides the touch points, a knowledge management framework is described to structure and share software security knowledge. Such frameworks clearly facilitate the spreading of knowledge and, hence, they are fundamental to education. Unfortunately, the book only hints at where and how this knowledge can be used throughout the different touch points.

Discussion: The common base-line for education is the training of project engineers in general and specific concepts related to software security. The training program of SDL is impressive in this respect, and some of these courses are even freely available. CLASP takes a somewhat different approach in that it broadens the target audience of training to all roles involved in the project. Touchpoints is more vague about the goal and the specifics of training. It is somewhat surprising, for instance, that given the risk-oriented approach of Touchpoints, no risk management training is suggested whatsoever. Finally, it is important to support and monitor the education program closely in order to optimize the net effect of training. In that respect, SDL suggests to institute a tracking program to keep track of who is required to follow which courses, as well as a measurement program to assess the effectiveness of knowledge transfer via training programs. In that sense, SDL puts more focus on the management perspective of education.

Awareness is a broad concept that can be interpreted in different ways. For SDL and Touchpoints, awareness relates to knowledge about the problem of software security, mostly by means of training and by clear commitment of the company's general management. In addition, CLASP also stresses the logistic side of awareness, for instance the sharing of project-specific security information to increase the awareness of engineers to the specific problems in the project at hand.

4.2. Project inception

SDL: At the beginning of this phase, SDL suggests to make a go/no go decision about the applicability of the methodology to the project at hand (Activity 2.1). If the gauge is passed, SDL describes in detail how to organize the personnel involved in a secure software project (Activities 2.2.1 and 2.2.2). Foremost, a security adviser is assigned to the project itself. This adviser helps the developers with security related issues and possibly serves as a gateway between developers and a dedicated security team (if available). Other important roles include the development team security contact, the company-wide security team and the security leadership team. While all the former are technical-oriented profiles, the responsibilities of the latter are more managerial and include (i) regular communication to the development team about security and privacy bug counts and (ii) communication of security and privacy policy updates.

Furthermore, SDL includes specific activities to address the logistics aspects, e.g., to make sure that the necessary tools are available (Activity 2.3), and to determine the type of security bugs that will (or will not) be addressed (Activity 2.4).

CLASP: For CLASP, the construction of the security team is also important. In this respect, CLASP suggests to *assign a security officer to the project* (Activity 2.2.2), which is a security expert that shares knowledge and acts as a security soundboard and reviewer throughout the development life-cycle. However, the influence of software security on other typical development roles (e.g., project manager, requirements specifier or architect) is discussed as well. CLASP also acknowledges the role played by individual commitment. To this aim, it suggests the use of both positive and negative incentives for motivating project engineers, by *institutionalizing accountability* (Activity 2.2.3) and by means of *rewards* (Activity 2.2.4).

An important difference in CLASP is represented by the attention that is put on *security metrics* (Activity 2.5), which aid in assessing the security posture of the product as well as enforcing accountability of security issues. Care must be taken to *identify the metrics to collect* (Activity 2.5.1), to *institute the strategy for data collection and reporting* (Activity 2.5.2), and to *periodically collect and evaluate the identified metrics* (Activity 2.5.3). Note that this last activity is actually ongoing during the different phases of the software development process.

Furthermore, CLASP emphasizes the importance of the *organizational policy* (Activity 2.6). It is suggested to develop such a document (if not already available), since it should be used as a baseline for security requirements of all software projects. CLASP provides a list of global security requirements to be used as a template for the global policy. During inception, the relevance of each global requirement must be evaluated for the project at hand. In case of conflicts between the global policy and project-specific requirements, proper action should be taken to resolve them.

Touchpoints: Touchpoints stresses the creation, and continuous execution, of an improvement program. This program aims at developing a clear understanding of the specific actions required to improve the state-of-practice within an organization regarding software security. The *improvement program* should cover both the *measurement strategy* (Activity 2.5.4) and the *process* itself (Activity 2.7).

Discussion: The processes cover a number of to-be-expected activities, which are also relevant for regular software development practices. For instance, one has to identify and assign responsibilities and, closely linked to that, to establish the positive and negative consequences for (not) meeting particular goals. Also, care must be taken to install program-wide, security-relevant logistic support such as tools for bug-tracking and team communication.

The support for security metrics is particularly important in making secure software construction a true engineering activity in which progress can be measured. Metrics are instrumental to assessing the quality of a project as well as the improvement over different projects. Examples of security-specific metrics include measuring the attack surface, the reported default rates and the coverage of security tests. In general, they can relate to different scopes: the project, the process, the product and the organization. Both CLASP and SDL thoroughly address the metrics-challenge and suggest a number of metrics that are related to particular activities. CLASP gives more guidance on which metrics to use and how to integrate them into the project life-cycle. For SDL on the other hand, given the elaborate and detailed description of many of the stages, it is not that hard to deduce extra, useful metrics (such as the number of threats covered, the number of bugs reported, or the test coverage). Touchpoints touches upon the use of metrics to drive the improvement program of an organization, but is more vague about how to realize this in practice. Unfortunately, none of the processes succeed in providing a complete and systematic integration of metrics within all process activities.

The project manager is the main stakeholder for most activities in this phase. He must make sure that all teams and all supporting infrastructure is in place and that basic agreements have been made. In this respect, one of the agreements that might deserve more attention in all processes is the overall security objectives that the project will target. For SDL, this is partially covered by the bug bar, but it would be useful to extend this idea to a broader number of indicators. Good agreements about this in the project inception phase renders the objectives more clear for all team members. Clearly, a prerequisite for this is the availability of a good set of indicators.

4.3. Analysis and requirements

SDL: This might be a striking observation, but SDL has very few activities in this phase. The definition of *use scenarios* (Activity 3.7) is part of the analysis of the system, and it is the first step in the (architecture-level) threat modeling process.

CLASP: In order to prepare for requirements specification, CLASP suggests to identify (i) *resources* (from a network as well as from a data perspective) and *trust boundaries* (Activities 3.1.1 and 3.1.2), (ii) *capabilities for resources* and link them to *roles* (Activities 3.2.2 and 3.2.1) and (iii) attacker profiles (Activity 3.3.1.1).

Requirements elicitation in CLASP is performed using both a black-hat and a white-hat perspective, i.e., by means of threat modeling and requirements specification respectively. Threat modeling uses different approaches: (i) *use case driven* threat modeling, during which attacks to all functional use cases of the system are performed (Activity 3.3.2.3), (ii) *resource driven* threat modeling that focuses on (il)legal use of resources (Activity 3.3.3) and (iii) *knowl-*

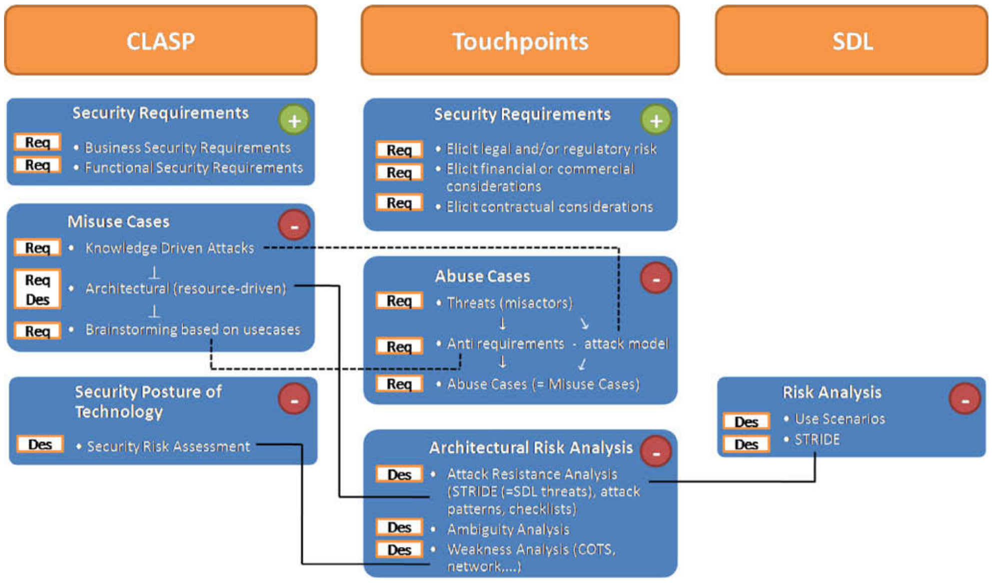

CLASP
Touchpoints
SDL
Security Requirements +
Req • Business Security Requirements
Req • Functional Security Requirements
Security Requirements +
Req • Elicit legal and/or regulatory risk
Req • Elicit financial or commercial considerations
Req • Elicit contractual considerations
Misuse Cases -
Req • Knowledge Driven Attacks
Req Des • Architectural (resource-driven)
Req • Brainstorming based on usecases
Abuse Cases -
Req • Threats (misactors)
Req • Anti requirements - attack model
Req • Abuse Cases (= Misuse Cases)
Security Posture of Technology -
Des • Security Risk Assessment
Risk Analysis -
Des • Use Scenarios
Des • STRIDE
Architectural Risk Analysis -
Des • Attack Resistance Analysis (STRIDE (=SDL threats), attack patterns, checklists)
Des • Ambiguity Analysis
Des • Weakness Analysis (COTS, network,...)

Fig. 2. Security requirements and threat modeling demystified.

edge driven threat modeling, i.e., assessing whether known attacks can be valid and useful for the system at hand (Activity 3.3.4.3). For each misuse case, appropriate *defense mechanisms* are to be identified (Activity 3.3.7).

Requirements specification is based on two pillars: *business requirements* and *functional security requirements* (Activities 3.4.4 and 3.4.5). The former covers all requirements that originate from the customer or the organization, possibly driven by the global security policy (see Section 4.2). The latter focuses on security services for the resources that were identified. A *conflict resolution* activity is suggested to solve mismatches between different (sets of) requirements (Activity 3.4.7). Finally, *risk mitigations* must be determined for each resource, similar to the defense mechanisms for misuse cases. In parallel, requirements for the *operational environment* are specified in order to improve the understanding of the system's context (Activity 3.6).

Touchpoints: One of the touch points is entirely dedicated to *threat modeling* based on abuse cases (Activity 3.3). The elicitation of abuse cases is achieved by identifying *threat agents*, eliciting *anti-requirements* (i.e., things the software should not do, often based on security functionality failure) and identifying the attack model (*attack patterns* relevant to the system at hand). Anti-requirements states what can go wrong (i.e., the goal of a possible break-in), while the attack model relates to the means to achieve this (the how). *Abuse cases* are then specified as the combination of a threat agent, an anti-requirement and an attack pattern, together with a mitigating scenario (Activity 3.3.5).

Apart from the aforementioned abuse cases, extra security requirements can be identified based on three different sources: laws and regulations, commercial considerations and contractual obligations (Activities 3.4.1, 3.4.2, and 3.4.3). All *security requirements are ordered* according to the severity of the obligation, which is among others related to the source of the requirement (e.g., not adhering to national laws may lead to severe business loss) (Activity 3.3.6).

Discussion: Threat modeling is one of the most fundamental activities in any security engineering effort. The spectrum of threat modeling activities that can be performed is considerable: conceptual versus technical, functionality- versus attack-driven, systematic versus pragmatic and so forth. Each process has its own accents in this spectrum and it is far from straightforward to understand the specifics of each activity and their relationship. Fig. 2 depicts our interpretation of this matter. For every process (represented in columns), the different activities are listed and lines denote a relationship between activities (a full line connects identical activities, while a dashed line represents a more loosely coupled relationship). Every activity is annotated to indicate whether it has to be executed during requirements analysis (Req) or software design (Des). Finally, activity groups are marked with a '+' for white-hat activities and a '-' for black-hat activities.

In general, it is fair to say that Touchpoints covers the broadest spectrum of activities, SDL is restricted to architectural threat modeling (see Section 4.4) and CLASP is positioned in between. In addition, it is also remarkable in this context that for all processes, resource-driven threat modeling is based on *technical* resources.[2] In the authors' opinion, it would make sense to identify business level resources (or *assets*) and use these to drive the threat modeling activities during the analysis phase. Indeed, assets do not necessarily align perfectly with technical ones, as is for instance the case for values that are typically not repre-

[2] While CLASP does not exclude the use of non-technical resources, the methodology is strongly biased and almost all the examples given in the text are system- or technology-oriented.

sented as first class concepts, but which are deduced by means of a query or some other computation (e.g., the global balance of a set of bank accounts). Consequently, the focus on business-level threat modeling would increase and business-level threats would be identified less accidentally. Touchpoints' risk management framework covers this to some extent, but its adoption is disconnected and not very visible in the touch points.

Another important goal of the analysis phase is the specification of security requirements. In this context, two observations are important. First, the sources used for the elicitation of threats or requirements are somewhat different. For SDL, this is primarily based on threats to the system. For CLASP, additional requirements originate from the global company policy, and functional security requirements are elicited explicitly. In Touchpoints additional requirements are based on obligations in the context of regulations or towards customers. This suggests that CLASP and Touchpoints use a broader set of sources to drive requirements elicitation. Second, it is noteworthy that abuse cases (or misuse cases) are a common format for the specification of security requirements. In the authors' opinion, this could be extended by explicitly transforming the misuses into more affirmative, solution-oriented requirements, which would align better with prevailing standards such as the Common Criteria ([5]).

As a final remark, the initial identification of mitigation strategies for threats can be useful, since this can lead to the elicitation of additional security requirements (in particular requirements on security functions). In the authors' opinion, the approaches for analysis-level mitigation described by the respective processes are weak. In CLASP, mitigation is based on *generic* knowledge of threats and, hence, only generic mitigation strategies can be suggested. In Touchpoints, mitigations should be specified in abuse cases but no guidance on how to achieve this is given whatsoever.

4.4. Architectural and detailed design

SDL: During the architectural phase, performing rigorous and thorough *threat modeling* (Activity 4.3) is possible as one has a clear understanding of the system decomposition and of the information flows within the architecture. SDL covers this by means of an extended set of sub-activities. Additionally, an interesting feature is represented by an activity focusing on the specification of the *operational environment* (Activity 4.3.3, also present in CLASP). This activity details the architectural assumptions about computing hosts and data networks.

At detailed design level, SDL focuses on assessing the impact of the project on user privacy (Activity 5.1) and the reduction of the *attack surface* (Activity 5.2). For instance, techniques are suggested to reduce the attack surface by discarding unnecessary features and by limiting privileges.

CLASP: Just like SDL, CLASP also supports threat modeling at architectural level, although in a less thorough way. However, CLASP adds two preparatory steps at the beginning of the architectural phase. First, CLASP considers the fact that most projects will be using third-party components. In order to prepare mitigating the security risks involved herein, CLASP includes activities to research and *assess off-the-shelf technology*, like third party components the project will depend upon (Activity 4.1). This activity is present in SDL as well, but in a limited form, i.e., the focus is on the interaction with the OS and the ActiveX controls. Second, CLASP is unique in suggesting a review step (Activity 4.2) where both the security and the non-security *requirements are audited* in order to assess their completeness.

At detailed design level, CLASP complements SDL in terms of activities devoted to the reduction of the attack surface (Activities 5.2.4–5.2.6). Indeed, SDL focuses more on reduction of privileges, while CLASP focuses on the *reduction of access points*.

Furthermore, CLASP is unique in providing guidance about injecting security *annotations in the design models* (Activity 5.3) and securing the *configuration of data bases* (Activity 5.4).

Touchpoints: The main focus of Touchpoints during the design phase is on threat modeling. For instance, Touchpoints includes *threat identification* and *risk assessment* (as described in the risk management framework), where risks in the system (including any frameworks that it might use) are identified and mitigated (Activity 4.3.8). Further, Touchpoints is unique in that it also aims at removing any ambiguity there might be, as ambiguity is often a cause of problems (Activity 4.3.9).

Discussion: As mentioned in the above description of this phase, all processes have a common trait regarding the analysis of threats at architectural level. By threat it is usually meant a danger for an architectural or infrastructural resource (playing the role of asset) like spoofing a server or intercepting a network connection. This type of threats take place "one level below" the application logic and are enabled by the inadvertent use of mechanisms like DNS for server spoofing or encryption for connection interception. It is interesting to note that all processes suggest the adoption of *misuse cases* (often referred to as threat scenarios) to document the possible threats to architectural elements. However, there are significant differences in the methodologies that the three processes suggest to use in order to unearth these threats. SDL is with no doubt the most precise and thorough. Indeed, the process suggests to use STRIDE, which is a systematic and effective way of eliciting threats (sometimes too effective, as the number of discovered threats can easily explode). Further, the technique is well documented and all the necessary resources to carry it out are provided, including a tool. Touchpoints is a bit less complete. It refers to STRIDE and proposes further activities related to threats. Examples are the analysis of weaknesses that arise by using third party software such as off-the-shelf components, and the analysis of flaws due to ambiguity in the design, for instance, problems related to type safety and type confusion. CLASP is deficient in this area, as it does not provide any guidance about how to carry out the threat analysis activity.

If we now look at the stakeholders involved in this phase, we notice some differences among the three processes. Both SDL and Touchpoints focus on the security expert that has to scrutinize the architecture. They also put significant emphasis on risk-driven prioritization of threats in order to allot development effort wisely. This allows the customers (who know the value of assets) and the project managers (who know the costs of development) to be involved in this process phase and to contribute with business-informed decisions. CLASP has a more holistic view and considers several stakeholders. During this phase, indeed, some activities link back to the requirements phase (review non-security requirements, assess completeness of security requirements) which is in line with modern software engineering approaches, where software architecture design and requirements definition are intertwined [6]. Therefore, requirements specifiers are involved in this phase too. Moreover, CLASP (as SDL) takes the deployment view into consideration, e.g., by validating the architecture and the design against both the network assumptions (e.g., existence of a firewall, of a centralized authentication server, and so on) and the host assumptions (e.g., expecting an important system component to be bundled with the OS). Clearly, this requires the involvement of field engineers. Finally, very detailed guidance is provided by CLASP to both the architect and the designer, e.g., with activities like *design hardened interfaces* and *annotate class design with security properties*.

4.5. Implementation and testing

SDL: It is interesting to observe that, apart from *secure coding guidelines* (Activity 6.5.1), SDL lacks real implementation activities. This is probably due to the fact that SDL focuses on security as a supporting quality. The use of *automated tools* for verification purposes is encouraged (Activity 6.3), as well as *manual code inspection* (Activity 6.4), e.g., to assess the adherence to coding guidelines. Furthermore, apart from the editing of *documentation* (Activity 6.7) that prescribe security best practices, the creation of *tools* (Activity 6.6) that can facilitate configuration and audit of systems is encouraged.

SDL puts a lot of focus on *security testing*. However, SDL focuses mainly on black box testing (Activities 7.1.10 and 7.1.11). A typical black box test is to feed the application with random input in order to observe the system's reaction for unexpected failures (which are hints of possible security bugs). SDL includes two additional testing activities: (i) the *security push* (Activity 7.2) and (ii) the *final security review* (Activity 7.3). Both perform extra testing but have a different focus. During the security push, the project team tests the entire system with specific focus on legacy code, while during the final security review the entire system is tested by the central security team. Both target the entire system with a focus on highest-risk components, in contrast to security testing which focuses on individual components. It is worth noting that, for the security push, SDL does cover white box verification in the form of code review.

CLASP: Just like SDL, CLASP emphasizes the importance of *security testing*. However, contrary to SDL, CLASP focuses more on white box testing (Activities 7.1.2 and 7.1.3). CLASP also suggests the integration of security analysis into source management (Activity 6.1), in order to automate the implementation-level security analysis and metrics collection through the use of dynamic and/or static analysis tools.

Other than these commonalities, CLASP includes more implementation activities, including the implementation of interface contracts (Activity 6.5.3). Moreover, CLASP acknowledges the importance of reviewing the specification from the developer perspective (Activity 6.2) in order to spot ambiguities that could lead to security flaws during implementation.

Finally, CLASP deals with the creation of the necessarily documentation to install and operate the application securely (Activity 6.8).

Touchpoints: Similar to SDL and CLASP, Touchpoints emphasizes the importance of *security testing* – actually, three out of seven touch points covered in the book deal with security testing. A difference in focus does exist, as Touchpoints stresses the importance of *risk based security testing* (Activity 7.1.1). The main characteristic of Touchpoints is the emphasis on code reviews. In particular, the use of automated tools is suggested (and examples provided).

Discussion: Not surprisingly, all analyzed processes stress the importance of security testing. A closer look to the documentation reveals that they all provide thorough, good-quality guidance in describing the testing-related activities. However, different flavors can be identified. SDL has a predominant black-hat approach to testing, i.e., activities focus on fuzz testing and penetration testing. CLASP is mostly white-hat, as illustrated by activities like resource driven testing, testing of security attributes (e.g., privileges), integration and automation of security testing with the commit procedure of the software repository and with the build process. Touchpoints is in between: some activities are mainly white-hat oriented, e.g., security functionality testing, while the black-hat component is still very present, for instance, pen testing is a touch point in its own.

Concerning the stakeholders that are involved in this phase, the implementers and the testers are center stage in all processes. However, we observe that SDL goes a bit further. Significant attention is devoted to provide the project managers with test results in order to track the project status (security-wise).

As a final consideration, we noticed that the use of both formal notations and the code generation techniques (e.g., MDA) do not find the proper space in any of the processes under study.

4.6. Release, deployment and support

SDL: SDL focuses on the *response plan* defining what to do when a new vulnerability is discovered (Activities 9.1 and 9.2). It is stressed that normal response planning can deviate considerably from the situation where a security emergency has to be dealt with. SDL also acknowledges that communication with customers is very important. One has to make sure that, amongst other things, security advisories can be released and customers have access to software updates.

CLASP: Even though organized quite differently from SDL, CLASP covers about the same activities, ending up with the same result. There is a difference though, and that is that CLASP puts extra emphasis on *signing the code*, in order to provide stakeholders with a way to validate the origin and integrity of the software (Activity 8.3).

Touchpoints: Touchpoints has very limited support in this phase. The only aspect that Touchpoints covers here is that the InfoSec people should contribute to the secure development by fine tuning access controls and configuring the monitoring and logging (Activity 8.4).

Discussion: As a general observation, we notice that the activities in this phase focus largely on support rather than deployment. As far as deployment is concerned, activities relate to the release of the software (e.g., code signing) rather than its fielding. Touchpoints is an exception in this respect, as it contains an activity dealing with the configuration of access control, logging, and monitoring. Concerning software support, two types of activities are very much elaborated in CLASP and SDL: (1) producing the documentation (for SDL triggered during the implementation phase) and (2) dealing with patching. However, some differences exist in the level of quality those activities are presented. Overall, CLASP does a good job in detailing the type of documentation to be produced. Quite interestingly, however, SDL acknowledges the importance of producing documentation also during the implementation and, therefore, has some activities in that phase too. With no doubt, SDL has a better coverage of the management of both security reports and patches. For instance, SDL prepares the patch management starting from the implementation phase (e.g., by easing up the bug fixing after release by uploading debug symbols) and follows up during the support phase with a well documented and structured set of activities.

5. Experimental assessment – the ATM case study

In this section, the application of the three processes on a small case study is discussed. The main purpose of this experiment is illustrative, i.e., to verify whether the results of the theoretical comparison hold in practice, rather than evaluative, i.e., to do a thorough and systematic analysis and comparison of the respective processes on the field. We first introduce the case study and then present the results of executing each process on the described case in Section 5.1 through Section 5.3. Each section concludes with a discussion part.

The case study is an ATM system, an internal project that is reused across several research works. The ATM software to be designed controls a simulated automated teller machine (ATM) having a magnetic stripe reader, a customer console, a slot for depositing envelopes, a dispenser for cash, a printer for printing customer receipts and a key-operated switch to start or stop the machine.

The main use cases of the system are to start or stop the system, and to perform a transaction (withdrawal, deposit, transfer or inquiry). The component diagram of the software architecture is shown in Fig. 3 (the process and deployment diagrams are very similar). The responsibility of most components is straightforward: at the terminal side, the ATMPhysical component abstracts the hardware of the machine, the Administrator- and CustomerInterface components represent the basic GUI for two different types of users, the ATMController contains the generic logic for the terminal, while the BankingLogic subcomponent focuses on the banking specific logic. At the BankingService side, the BankingService component contains the logic for the bank, while the LoggingSystem is responsible for logging relevant events.

During the experiment, the three processes were applied to this case study based on the available material. From a methodological perspective, each process was applied in a project-oriented way by building a team of two persons selected from the authors. Each team was composed of a junior researcher and a senior researcher acting as a soundboard. At the end of the experiments, the results were reviewed during a workshop in order to ensure completeness.

The case study is developed to the level of architectural design. Therefore, only a subset of the process phases can be applied. As motivated by the above description, the "Education and awareness" and the "Project Inception" phases were bypassed. Indeed, the creation of the teams was straightforward due to the resource constraints, and the training of team members in the security area was sufficient. Furthermore, these phases are of less use for one-shot projects. Since no implementation of the case is available, the core set of activities executed reside in the "Analysis and Requirements" and "Architectural Design" phases. This implies that some important activities, like testing, were not executed.

5.1. CLASP

5.1.1. Analysis

In the context of analysis, the first activity is to determine resources and trust boundaries (Activity 3.1). In the case study, the network design (Activity 3.1.1) was based on the architectural deployment diagram, except that the

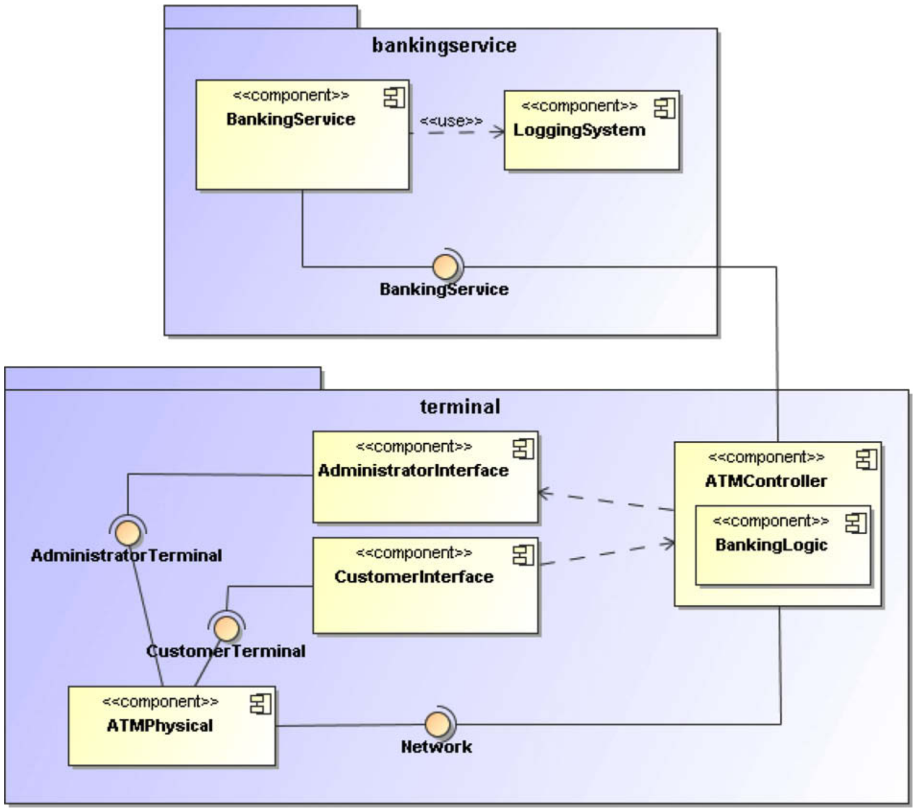


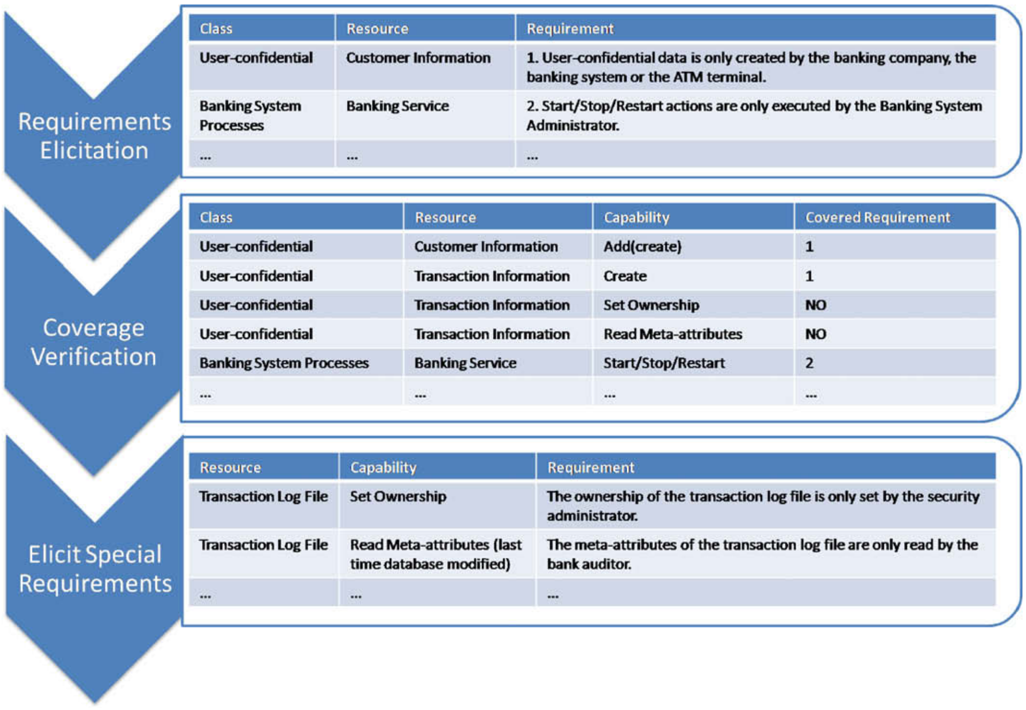
Fig. 3. Architectural overview of the ATM case study.

components of the banking service were distributed over different machines (this enabled more flexible deployment scenarios and increases as such the depth of the security analysis). To identify data resources (Activity 3.1.2), CLASP suggests to look for several items, like: service level access points and information assets, low level resources (such as memory), and system related resources (such as ACLs and configuration files). At this point of the development life-cycle only the identification of the first type was considered really feasible (and useful). Since many activities of CLASP are driven by resources, extra consideration was put into these results, but no satisfying improvements could be achieved.

Resources are then used in two ways by CLASP: white-hat and black-hat analysis. During the white-hat part, security requirements are elicited for the resources. In the black-hat part, attacks are matched to the identified resources.

For the white-hat track, the capabilities of the resources have been identified (Activity 3.2.1), and system roles have been mapped to these capabilities (Activity 3.2.2). This is instrumental to the following step, i.e., the elicitation of security-relevant requirements (Activity 3.4.5). During this activity, the resources have been first grouped into five broad classes: user-confidential data, ATM processes, bank system processes, network and memory. The choice of the classes is arbitrary, however, the rationale is to minimize the number of requirements that have to be specified by bracketing resources that have similar demands. In our experiment, 12 general requirements have been specified for the 5 classes. In a next step of the activity, the coverage of the requirements has been verified by mapping them onto individual resource capabilities and assessing whether extra, specific requirements would be necessary (gap analysis). In a final step, these extra requirements have been articulated for the individual resource capabilities. An overview of this process and an excerpt of the results are shown in Fig. 4.

For the black-hat track, the next step is the identification of the threat agents (Activity 3.3.1.1). As a result of this activity, 5 profiles have been described: insider (e.g., an administrator or a bank employee in charge of the ATM), script kiddie (e.g., to get easy money), competitor (e.g., to gain customers information), government (e.g., to track banking transactions), and organized crime (e.g., to cover transactions). An additional profile (activist) has been considered and discarded. The resources and the profiles are the input to the subsequent threat modeling activities. The identification of misuse cases is performed in three ways: based on known attack patterns, on functionality (i.e., use cases), and on identified resources. For each of these, a number of misuse cases have been identified. One of the problems that we have encountered in this context is the generation of overlapping misuse cases over the three approaches: the lists had to be sanitized considerably.


Requirements Elicitation
Coverage Verification
Elicit Special Requirements

Class	Resource	Requirement
User-confidential	Customer Information	1. User-confidential data is only created by the banking company, the banking system or the ATM terminal.
Banking System Processes	Banking Service	2. Start/Stop/Restart actions are only executed by the Banking System Administrator.
...	...	...

Class	Resource	Capability	Covered Requirement
User-confidential	Customer Information	Add(create)	1
User-confidential	Transaction Information	Create	1
User-confidential	Transaction Information	Set Ownership	NO
User-confidential	Transaction Information	Read Meta-attributes	NO
Banking System Processes	Banking Service	Start/Stop/Restart	2
...	...	...	...

Resource	Capability	Requirement
Transaction Log File	Set Ownership	The ownership of the transaction log file is only set by the security administrator.
Transaction Log File	Read Meta-attributes (last time database modified)	The meta-attributes of the transaction log file are only read by the bank auditor.
...	...	...

Fig. 4. Defining requirements in CLASP.

5.1.2. Design

At architectural level, there is one main activity that has been applied to the case study: identification of architectural level misuse cases (Activity 4.3.8). The method is comparable to Microsoft STRIDE/DREAD. However it is less structured: SDL provides threat tree patterns that one can use in practice, while the CLASP resources about vulnerability causes is more limited.

Other activities like assessment of third party components (Activity 4.1) and the review of requirements (Activity 4.2) have been omitted because they are not fundamental in the specific context of this case study.

5.1.3. Discussion

Overall, CLASP was perceived as a systematic method, although a number of activities can be improved in this respect (e.g., the identification of resources, or architectural-level misuse cases). The execution of the process took more time than expected: over 40 h, with the identification of requirements as the most demanding activity in this respect. The identification of requirements was perceived as the most difficult activity, probably due to the fact that it is hard to come up with requirements without proper (guidance for) motivation.

5.2. SDL

5.2.1. Analysis

As discussed in Section 4.3, SDL has very few activities in the analysis phase. The definition of use scenarios is related to threat modeling, which will be discussed in the next section.

5.2.2. Design

Product risk assessment (Activity 4.1) is concerned with identifying the risk involved in the application. The questionnaire used for security risk assessment did not reveal any additional information, especially since at the time of execution many of the questions were not applicable to the initial architecture of the system yet. It would be more useful if the architecture and the design of the case would have been more worked out.

For threat modeling (Activity 4.3), the first step was to create data flow diagrams (DFD) for the system (Fig. 5 shows one of these – the colored ovals indicate which STRIDE threats are applicable to a particular entity). Thereafter, threats were to be identified by applying STRIDE on all entities in the DFD. Microsoft provides a nice threat modeling and analysis tool [7] to aid in this effort. While the book and the tool do not fully correspond (a.o. for threat elicitation and risk assignment), the tool

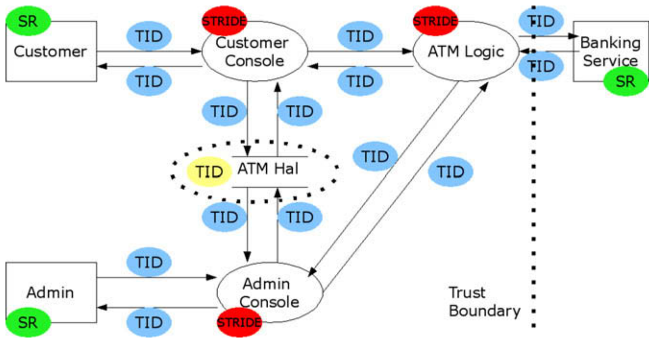


Fig. 5. DFD for the customer side of the ATM system.

provides a good help in identifying, automating and analyzing the threats to a system. For this case, 118 threats were identified. In terms of risk analysis, the book suggests a new approach by assigning risk levels depending on the characteristics of threats (e.g., in the context of data tampering, is the effect temporary or permanent), rather than assigning values for a threat's likelihood and impact, since this is typically perceived as a difficult task. For the experiment, most of the threats were identified as level 2, while the other levels (1, 3 and 4) were assigned only sporadically.

5.2.3. Discussion

Overall, the majority of time was spent on the threat modeling (approx. 30 h). Threat modeling was perceived as a feasible activity, except for the rating of risks, which is rather arbitrary.

One of the clear benefits of SDL is the systematic approach to the architectural threat modeling. By assigning STRIDE threats to all elements in the data flow diagrams, no obvious threats are missed. The other processes by far do not perform equally well for basic threat modeling. However, there are some downsides linked to the STRIDE threat modeling as well: the systematic analysis results in an explosion of elicited threats, which then requires a good risk analysis method to be managed. Unfortunately, the risk assessment technique does not seem to perform very well for our case. Most threats were assigned risk level 2. This can be explained by the fact that the characteristics are not always fully clear when applied to practical situations, or the fact that the risks to be assigned are not very well spread over the risk spectrum (more than half of the leaves in the threat tree patterns assign risk level 2). Another downside to the process is that it focuses on elements in isolation, rather than identifying threats to a particular combination of elements. The latter might reduce the explosion of threats, since many threats are actually linked to a few higher-level threats (analogous to the idea of threat trees).

5.3. Touchpoints

5.3.1. Analysis

According to the McGraw's theoretical model, three touch points are of interest in this phase: Security Requirements, Risk Analysis, and Abuse Cases.

In particular, the *security requirements touch point* suggests to elicit financial or commercial considerations, contractual obligations, and legal/regulatory risks (see Activities 3.4.1–3.4.3). We applied this activity to the banking domain with the following results:

- Financial and commercial considerations. Commercial assets such as reputation are important in this domain. Part of keeping up a good reputation is making sure the system is "always" available as well as limiting the number of incorrectly executed transactions (availability requirement). Financial considerations are also important. For example, it was recognized that system has to assure that the transactions are executed using the correct amount of money involved, i.e., no rounding errors and such are introduced (accuracy requirement).
- Contractual obligations. As the client stakeholder is not available in our case, we have no results out of this activity.
- Legal and regulatory risks. The system deals with personal information, hence privacy laws must be taken into account. Other legal implications concern the obligation of keeping track of dodgy/fraudulent behavior for judiciary purposes (auditing requirement). Further, since transactions might constitute as evidence used in court, repudiation of such transactions should not be possible (non-repudiation requirement). In particular, supposing that the system is to be deployed in Europe, four Council Directives apply.

The *abuse cases touch point* proved to be very powerful. In this area, three main activities were performed:

- Identify and Revise Threat agents (Activity 3.3.1). Five types of attacker profiles were identified: organized crime, small criminals, insiders, legitimate users, script kiddies.
- Use Case Driven Analysis (Activity 3.3.2). A list of 20 scenarios that the system should prevent were identified, such as retrieving more money than is available in the account, depositing money to an account without identification, or shutting down the ATM system without authorization.
- Attack Driven Analysis (Activity 3.3.4). From a list of attack patterns provided by the process as a resource, 21 attacks were selected as relevant for the case study.

It is interesting to note that each activity must be performed iteratively. Each activity is followed by a review step until the quality is satisfactory and the results are approved.

In Activity 3.3.4, the abuse case are finally produced as a cross-product of threat agents, anti-requirements, and attacks. Of course, not all attacks can be performed by each attacker profile. For instance, complex attacks are not considered for script kiddies. Nevertheless, we observed an explosion in the number of abuse cases. The output is schematized in Fig. 6, which, due to space constraints, does not include the complete results.

In the final step, the abuse cases were ranked according to risk (Activity 3.3.6). We applied a three-level categorization: must-have, important to have, nice but unnecessary to have. This classification is suggested by the *risk analysis touch point* when it comes to ranking the security requirements (described above). For the sake of homogeneity, we used the same ranking criteria for abuse cases as well. As a result of this activity, out of the 35 identified abuse cases, 74% were ranked as must-haves, 23% as important, and only 3% as unnecessary.

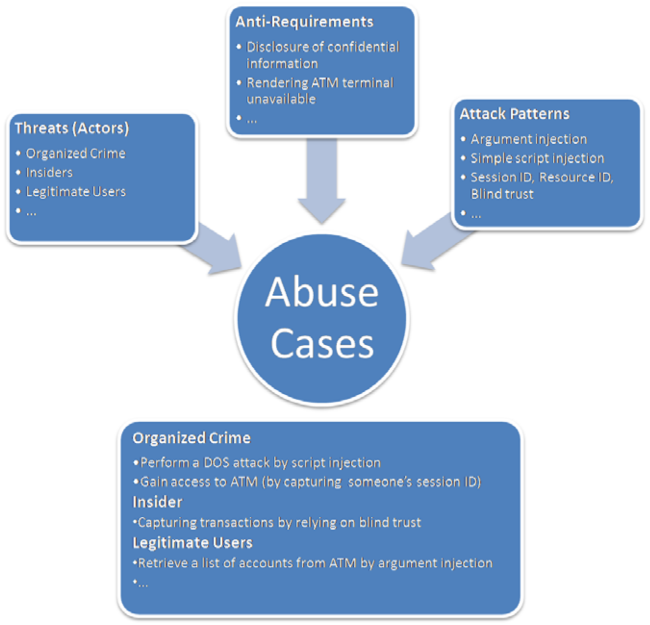

Anti-Requirements
• Disclosure of confidential information
• Rendering ATM terminal unavailable
• ...
Threats (Actors)
• Organized Crime
• Insiders
• Legitimate Users
• ...
Attack Patterns
• Argument injection
• Simple script injection
• Session ID, Resource ID, Blind trust
• ...
Abuse Cases
Organized Crime
•Perform a DOS attack by script injection
•Gain access to ATM (by capturing someone's session ID)
Insider
•Capturing transactions by relying on blind trust
Legitimate Users
•Retrieve a list of accounts from ATM by argument injection
•...

Fig. 6. Generating abuse cases in Touchpoints.

5.3.2. Design

The design phase is covered by the *architectural risk analysis touch point*. This phase is less rich of distinguishing results. Indeed, architectural risk analysis is largely based on the STRIDE technique, which has been already illustrated in Section 5.2.

As far as Activity 4.1.1 is concerned, it was skipped as no third party component is used in this simple application.

5.3.3. Discussion

Overall, the time needed to carry out the experiment was equally distributed among the analysis phase (14 h) and the design phase (14 h). The creation of the abuse cases was perceived as a low-to-medium complexity task; the architectural risk analysis as hard, especially the ambiguity and weakness analysis.

In conclusion, some general observations have been gathered during the execution of this experiment. On the positive side, the team was particularly pleased by the thoroughness of the analysis phase. First, there is some focus on laws and regulations. In several sectors, like health, aviation and finance, the existing regulations and laws are indeed primary sources of (security) requirements. Second, the methodological way of building high-level anti-requirements (via abuse cases) is particularly interesting. McGraw's methodology acknowledges the importance of having two levels of misuse analysis, i.e., both at the requirements and the architectural level.

On the negative side, applying the touch points is not always straightforward. Both the Abuse Cases and the Architectural Risk Analysis are described via a detailed flow chart of activities. This has been appreciated by the team, however, a limited amount of information about *how* to perform each step in the chart is provided. In other words, the team found that too often there is room for interpretation and uncertainty about what to do in practical terms. Second, the process tends to get too technical too soon. An example is the list of attacks provided during analysis in order to elicit the abuse case. For instance, during the requirements phase, it is hard to decide whether a "buffer overflow with environment variables" is a viable attack, as the detailed design description of the system is yet to come.

6. Discussion and improvements

In this section we discuss a number of fundamental improvements, which the processes could benefit from.

6.1. Quality and coverage

During the analysis of activities, it was observed that some of them do not correspond to the state-of-the-practice definition for process activities. From a general perspective, a process involves a temporally ordered series of activities that, starting from an input state, leads to an outcome by using a set of resources like time, and expertise.

Activities must produce added value in the form of an output, which must be clearly identified in an artifact. As a counter-example, only two activities in CLASP clearly specify the artifacts that should be produced. Touchpoints scores better here as for some touch points they provide a process diagram stating the outputs of the different activities. It is also important that the delta between input and output, the *'visible impact'*, is large enough to make an activity worthwhile. If this criterion is applied to the processes, it can be observed that some of the construction steps are guidelines rather than real activities. For instance, SDL suggests to use the latest compilers and to avoid using deprecated functions from libraries. According to the above discussion, this is more of a coding guideline than a process activity. A similar but reverse example comes from the Touchpoints. An initial step of the static code analysis activities is the selection of the tools to be used for the automated analysis itself. In that case, the impact is more visible, as the activity starts from a list of candidates and ends up with a set of selected tools and their associated configuration (to be used later on in the process). As another positive example, CLASP addresses the impact of each of the 24 top-level activities in the Activity-Assessment View.

According to Davenport [8], the characteristic of a business process is to focus on the procedural aspects of the activities, i.e., how work is done and not only what work has to be done. That is, activities should provide a *'systematic method'* to put in practice what they suggest. It is observed that SDL and CLASP both fall short in supporting the aforementioned property. For instance, the CLASP activity *apply security principles to design* (Activity 4.4) has no clear methodological indication on how to realize the activity. McGraw's process provides a (basic) method for some of the touch points, but unfortunately not for all of them. This method is shown in the form of a process diagram and some related information. A good example is the diagram explaining the steps to perform the weaknesses analysis of off-the-shelf components during the architectural risk analysis touch point (see Activity 4.4.13).

Besides these quality issues, a few coverage gaps were observed, i.e., phases that have little or no support. First, activities at the design stage are mostly oriented towards detailed design. For instance, at architectural level, there is room for specific activities in the area of trade-off analysis, e.g., security vs. usability. Second, in both SDL and Touchpoints there is very limited support during the deployment phase. In SDL it is limited to user documentation, while Touchpoints only talks about the contribution that InfoSec people can bring into the project at that phase (Activity 8.4). Further support for deployment could include the management of the security solution by enforcing operational procedures, as well as the monitoring of the security solution in order to proactively discover weaknesses.

6.2. Security in context

Two main constraints must be considered when it comes to security-specific activities: the integration with an existing process and the cost of augmenting a process with security.

Concerning *integration*, many medium-to-large organizations already have a heavyweight and rigorous development process in place, e.g., according to the Unified Process [9]. Furthermore, many of them have achieved a significant level of process maturity and have received a certification for their accomplishment. For these organizations, it is key to have a set of clear guidelines to ease up the difficult task of integrating new security-centric activities in a well-established, preexisting instance of a standard process.

For smaller organizations with a more lightweight and less rigorous process, merging the security process with the existing informal software development process is less of a problem. However, these organizations may have a non-traditional process in place, e.g., they may be using agile techniques such as Extreme Programming [10]. There is a knowledge gap with respect to the problem of integrating CLASP in agile methods. Conversely, SDL's and Touchpoints's support in this direction is far more extensive.

It is impossible to build a 100% secure system, especially in day-to-day system construction with a constant trade-off between security and other constraints such as the *cost*. This will have an impact on the accuracy with which certain activities are being executed and on the set of activities that will be executed in the first place. Therefore, it would be useful to provide guidelines for selecting the set of relevant activities. At a minimum, a distinction could be made between a core set of mandatory activities and its accessory extensions. Also, an indication of the risk associated with the omission of accessory activities is useful. For instance, CLASP provides such information to some extent (in the Activity-Assessment View) and Touchpoints has a prioritization of the touch points that can be used as a guide. It would seem useful to extend this into a maturity-like approach (e.g., based on SSE-CMM [11]) in order to indicate the desired rigor to be put in the execution of certain activities more precisely. Based on this, a number of profiles could be defined for different types of environments and applications domains. Additionally, these indications could be used to drive the assurance process of the software afterwards.

6.3. Verification

Application functionality is a positively spaced problem: functional requirements are specified in an affirmative manner (what behavior should occur) and verifying the correctness of a functional requirement focuses on the (limited) set of executions of the particular function. In clear contrast, security is to a large extent negatively spaced. Security requirements typically state which behavior should not occur. Since the set of executions of a software artifact is infinite, it is much harder to cover this set in order to guarantee the correct realization of a security requirement in software. Therefore, it is crucial to institute verification as an important fill rouge throughout the process, complementary to construction activities.

Verification is in practice often closely linked to a particular construction activity, i.e. to ensure that the activity has been done right. For instance, when building a threat tree one could verify that all types of threats are covered systematically for every asset in the system. While the importance of this type of verification is clear, it is equally important to include dedicated verification efforts that focus on relations between activities as well as on spanning significant sets of activities. For instance, one could cross-check the security requirements set with the operational policy.

The experiment with the ATM system has shown that the verification of results of activities is difficult in practice since (i) it is often not clear what it means to be correct (i.e., no precise criteria have been defined) and (ii) the method to be used for verification is not defined. Consequently, most verification has been performed manually and in an ad-hoc manner. There is a lot of room for improvement in this area. Clearly, the availability of good metrics (see next section) can be a catalyst for improving verification.

6.4. Metrics

The software engineering practice assigns a significant role to metrics. They are a quantitative means to measure the progress of quality for the process itself and the produced artifacts. CLASP, Touchpoints and (particularly) SDL all have only limited focus on metrics. Two directions of improvement can be foreseen.

First, a program to evaluate the effectiveness of the security process itself must be instituted. The program should be responsible for continuously assessing both the real impact of security activities on product quality and the optimal usage of resources (e.g., time and personnel). The importance of measuring the impact of security is somewhat mentioned by examples in Touchpoints, but the metrics program devised by McGraw is primarily intended to measure the level of adoption of the touch points, i.e., it is a CMM-like measurement program. To assess the optimal usage of the security budget, the process should provide baseline figures about time and personnel required by the activities. For instance, CLASP provides information about the time that is required to carry out activities.

Second, specific measuring activities should be included in all development phases to assess the quality of artifacts from a security perspective and at different levels of abstraction. Such measures could play a leading role in defining acceptance criteria for software artifacts during all stages of the development process. More importantly, metrics should constitute an analysis tool to identify criticalities early on, with remarkable impact on cost. To some extent, CLASP tries to pursue this direction, as a whole top-level activity is dedicated to metrics. However, the implementation of this type of measuring program is difficult. Indeed, there is not a clear understanding of which properties should be observed in order to asses security qualities. Some work exists concerning the implementation and the deployment phase [12], but significant gaps have to be filled by the research community.

6.5. Principled process

Several security principles have been enumerated in the literature [13–15]. Examples include 'minimize attack surface' and 'run with least privilege'. The value of such principles is so widely recognized that their enforcement should be explicitly assisted by the process.

Activities could be annotated in order to highlight their contribution to the realization of a specific principle. For instance, CLASP activity *map roles and resources to entry point* (Activities 5.2.5 and 5.2.6) relates to the 'minimize attack surface' principle. In CLASP, principles are seldom addressed a few times, but the relationship between activities and principles is not worked out systematically. In SDL, principles are explicitly mentioned only twice. In Touchpoints they are explicitly mentioned as part of the knowledge pillar of the process, i.e., the part of the method that covers the sources of information to be used during the activities. However, there is no explicit mention about how to use them practically in the touch points. Furthermore, new constructive activities could be introduced to better support principles. Finally, some verification activities can be put in place to assess the compliance of process artifacts with principles. For instance, the SDL activity *Scrub the attack surface* (Activity 5.2.7) contributes to the verification of the above-mentioned 'minimize attack surface' principle.

6.6. Process support for moving targets

It is common knowledge that security is a moving target: applications change, execution environments change, attackers change, and new (types of) bugs are found. This influences the security posture of a software application and, consequently, it also affects the construction process of secure software in different ways. First, under the hypothesis that the previously articulated security assumptions remain valid, the process must include continuous support to address new security vulnerabilities after the software has been released. This is supported in CLASP and SDL by including activities that focus on software updates and security advisories. Touchpoints does not seem to cover this. Second, and more challenging, when intermediate results turn out to be incorrect (such as an incomplete threat model), or when security assumptions change after deployment, the process must be backtracked in order to correct the no-longer valid decisions and assumptions. In a process, backtracking can be supported by introducing iterative cycles, or by inserting dedicated checkpoints and feedback loops. This kind of support is limited in the covered processes.

In the same way, it is challenging to enforce security requirements on a moving target, functionality-wise. For instance, if the use cases of a system change frequently, threat modeling must be repeated every time and since many activities are based on the outcome of threat modeling, this will have several ripple effects. These effects could be limited by choosing a limited set of locations in the process to which security activities are added.

Traceability of decisions and relations between intermediate results constitute a way to improve the support for change management. As such, one can more clearly analyze and estimate the impact of modification on the developed system.

7. Related work

To the authors' knowledge, no evaluative studies have been performed on secure software development processes. Therefore, the discussion on related work focuses on (i) related processes covering the entire secure development life-cycle, and (ii) methodologies focusing specifically on particular construction or assurance activities.

There are a number of resources that cover the entire secure development life-cycle. Build Security In [16] is a website maintained by the Software Engineering Institute (SEI) and sponsored by the DHS National Cyber Security Division. Part of the available material and their contributors are closely related to Touchpoints. The website provides an overview of existing processes and methods for each development phase. These methods are not secure software methodologies, but rather tools that can be used and that only cover part of a secure software engineering methodology. In his articles, Peterson [17] discusses how a development team can integrate security into any iterative development process by finding what (artifacts) already exists and how this can be leveraged for security's goals and by whom. Praxis [18] proposes a methodology to build reliable and secure software by ensuring correctness for every step of a software life-cycle process, often by using formal verification techniques. The Information Assurance Technology Analysis Center presents an extensive overview of security-enhanced processes [19], including among others the Oracle Software Security Assurance Process and TSP Secure.

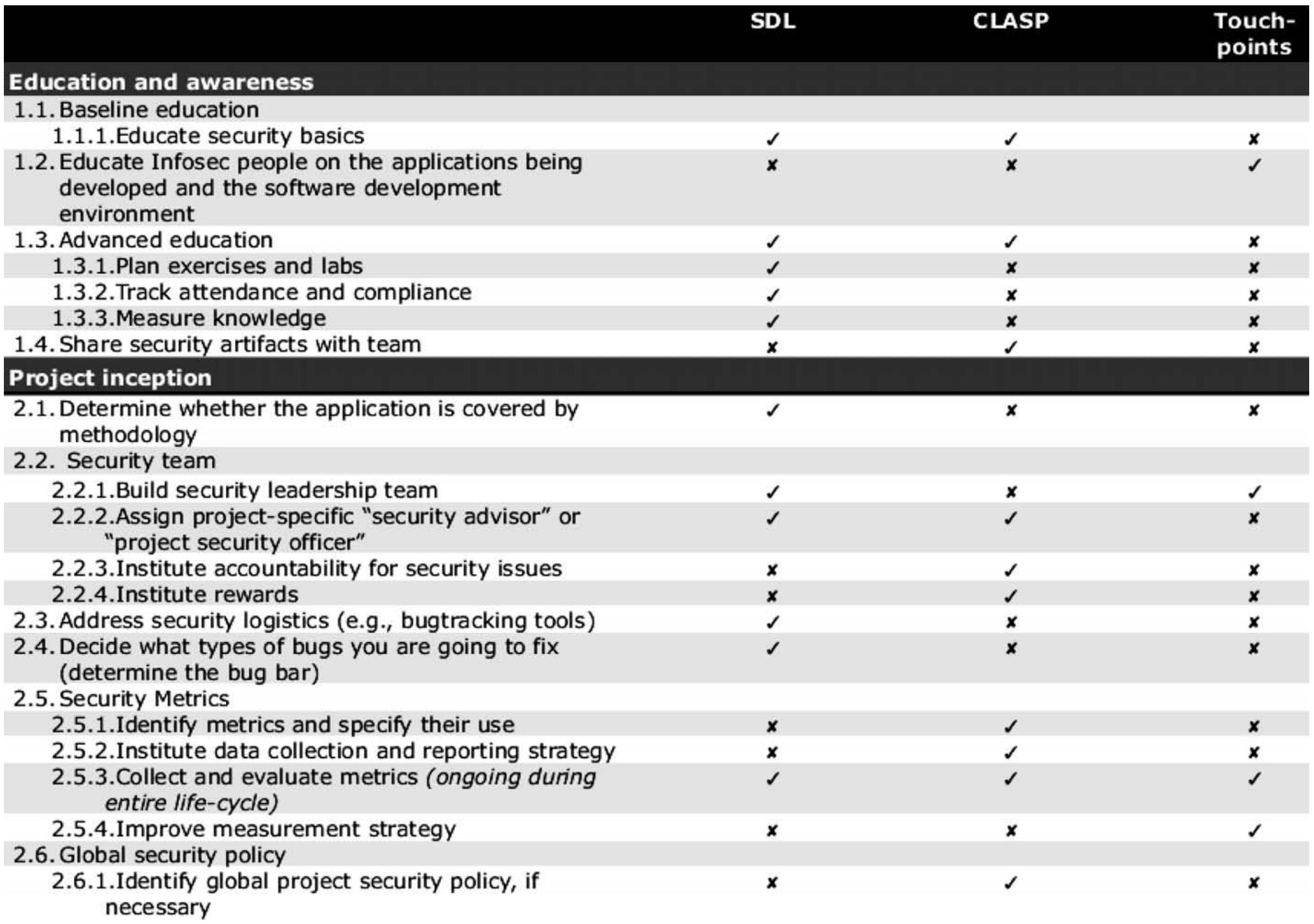
A wealth of other sources either address particular phases of the development life-cycle, or focus on specific platforms. We only highlight a few. Howard and Leblanc [20] elaborate on designing secure applications, on writing robust code and on testing applications. Wheeler [21] provides a set of guidelines for designing and implementing secure programs on the Linux and Unix platforms.

Concerning methodologies for assurance, the Common Criteria provides assurance that the process of specification, implementation and evaluation of a product has been conducted in a rigorous and standard manner. Similarly, the Systems Security Engineering Capability Maturity Model (SSE-CMM) [11] covers all phases of the development process and can be used to evaluate and improve an existing process.

8. Conclusions

A key challenge for scientists in software engineering in the coming years is to evolve the 'art' of building secure software into a systematic and manageable practice. The problems in this area are caused by a number of reasons, including the continuous streams of novel security issues (and solutions to address them), as well as the difficulty of linking vulnerabilities to particular poor practices or intermediate results in the construction process. Secure software processes are forced to target on specific subproblems in this space and, hence, perform very well for some types of problems and less for others. Consequently, a comparison of the strengths and weaknesses of processes for secure software development is beneficial to all software practitioners.

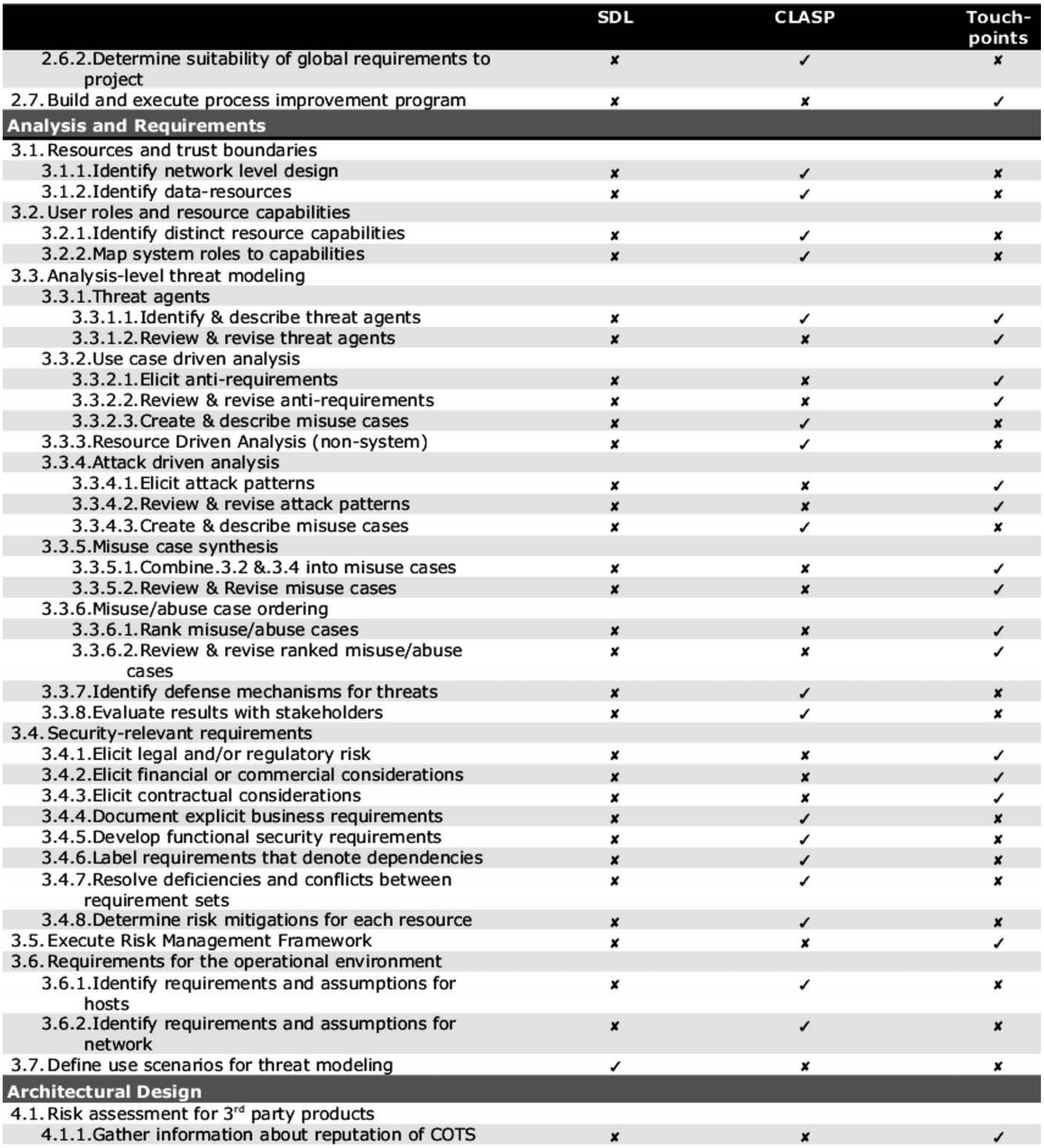
This paper has evaluated and compared three forefront processes for secure software construction: Microsoft's SDL, OWASP's CLASP and McGraw's Touchpoints. As



	SDL	CLASP	Touch-points
Education and awareness			
1.1. Baseline education			
1.1.1. Educate security basics	✓	✓	✗
1.2. Educate Infosec people on the applications being developed and the software development environment	✗	✗	✓
1.3. Advanced education	✓	✓	✗
1.3.1. Plan exercises and labs	✓	✗	✗
1.3.2. Track attendance and compliance	✓	✗	✗
1.3.3. Measure knowledge	✓	✗	✗
1.4. Share security artifacts with team	✗	✓	✗
Project inception			
2.1. Determine whether the application is covered by methodology	✓	✗	✗
2.2. Security team			
2.2.1. Build security leadership team	✓	✗	✓
2.2.2. Assign project-specific "security advisor" or "project security officer"	✓	✓	✗
2.2.3. Institute accountability for security issues	✗	✓	✗
2.2.4. Institute rewards	✗	✓	✗
2.3. Address security logistics (e.g., bugtracking tools)	✓	✗	✗
2.4. Decide what types of bugs you are going to fix (determine the bug bar)	✓	✗	✗
2.5. Security Metrics			
2.5.1. Identify metrics and specify their use	✗	✓	✗
2.5.2. Institute data collection and reporting strategy	✗	✓	✗
2.5.3. Collect and evaluate metrics *(ongoing during entire life-cycle)*	✓	✓	✓
2.5.4. Improve measurement strategy	✗	✗	✓
2.6. Global security policy			
2.6.1. Identify global project security policy, if necessary	✗	✓	✗

Fig. A.1. Matrix: part I of V.

a first major contribution, the paper has articulated and structured the processes into activities, and *demystified the relationships* between the different processes. The complexity of this should not be underestimated, as illustrated, for instance, by the overlap among the different techniques used during the elicitation of requirements and threats. A second important contribution is the *identification of key differentiators* of the different processes. For instance, it has become clear that SDL is very thorough in architectural threat analysis, Touchpoints provides a systematic way of eliciting analysis-level threats and CLASP includes the best support for architectural design. Also, the particular focus on, and different balance between, white-hat and black hat security testing in the different processes is an

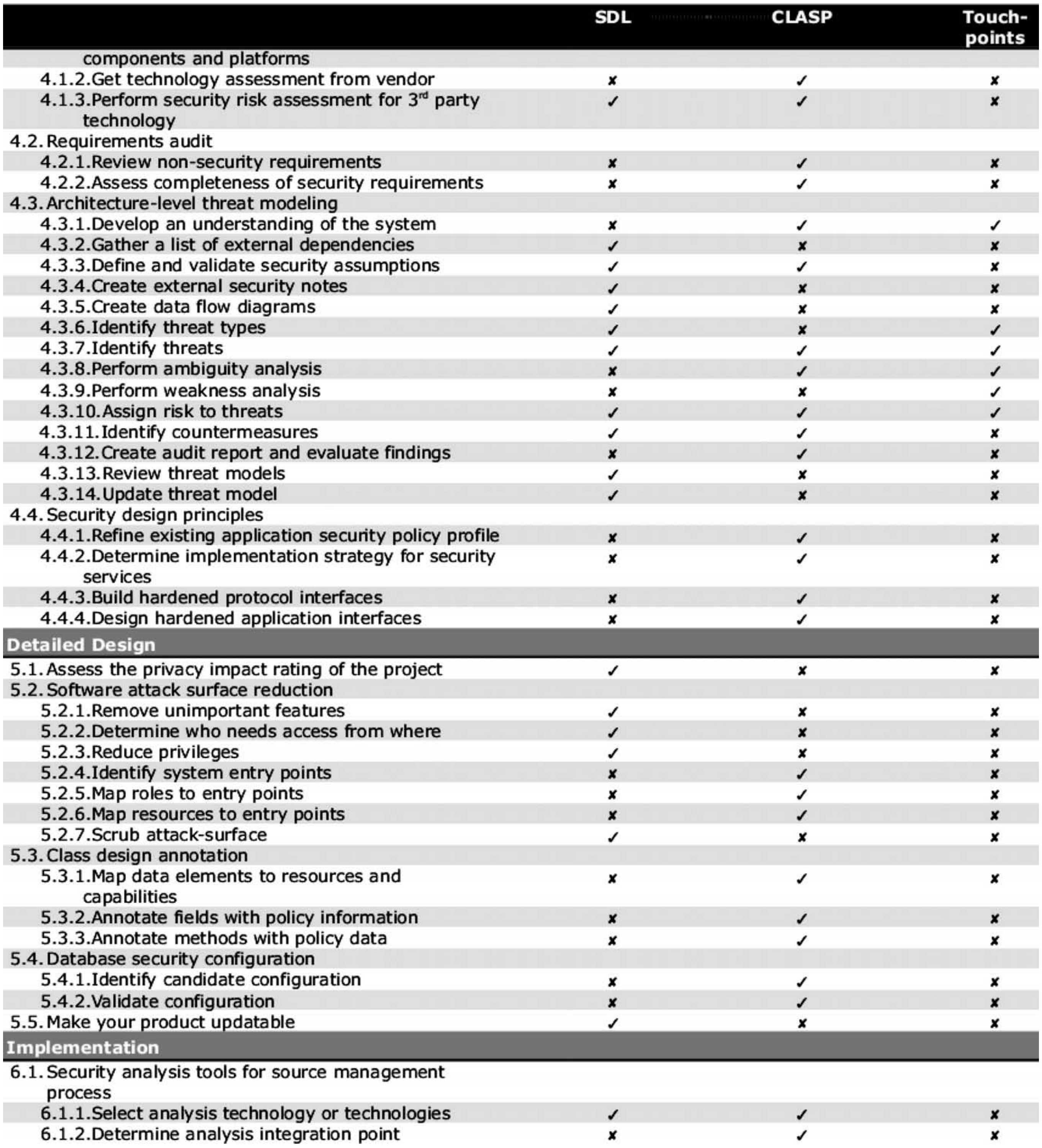


	SDL	CLASP	Touch-points
2.6.2. Determine suitability of global requirements to project	✘	✓	✘
2.7. Build and execute process improvement program	✘	✘	✓
Analysis and Requirements			
3.1. Resources and trust boundaries			
3.1.1. Identify network level design	✘	✓	✘
3.1.2. Identify data-resources	✘	✓	✘
3.2. User roles and resource capabilities			
3.2.1. Identify distinct resource capabilities	✘	✓	✘
3.2.2. Map system roles to capabilities	✘	✓	✘
3.3. Analysis-level threat modeling			
3.3.1. Threat agents			
3.3.1.1. Identify & describe threat agents	✘	✓	✓
3.3.1.2. Review & revise threat agents	✘	✘	✓
3.3.2. Use case driven analysis			
3.3.2.1. Elicit anti-requirements	✘	✘	✓
3.3.2.2. Review & revise anti-requirements	✘	✘	✓
3.3.2.3. Create & describe misuse cases	✘	✓	✘
3.3.3. Resource Driven Analysis (non-system)	✘	✓	✘
3.3.4. Attack driven analysis			
3.3.4.1. Elicit attack patterns	✘	✘	✓
3.3.4.2. Review & revise attack patterns	✘	✘	✓
3.3.4.3. Create & describe misuse cases	✘	✓	✘
3.3.5. Misuse case synthesis			
3.3.5.1. Combine.3.2 &.3.4 into misuse cases	✘	✘	✓
3.3.5.2. Review & Revise misuse cases	✘	✘	✓
3.3.6. Misuse/abuse case ordering			
3.3.6.1. Rank misuse/abuse cases	✘	✘	✓
3.3.6.2. Review & revise ranked misuse/abuse cases	✘	✘	✓
3.3.7. Identify defense mechanisms for threats	✘	✓	✘
3.3.8. Evaluate results with stakeholders	✘	✓	✘
3.4. Security-relevant requirements			
3.4.1. Elicit legal and/or regulatory risk	✘	✘	✓
3.4.2. Elicit financial or commercial considerations	✘	✘	✓
3.4.3. Elicit contractual considerations	✘	✘	✓
3.4.4. Document explicit business requirements	✘	✓	✘
3.4.5. Develop functional security requirements	✘	✓	✘
3.4.6. Label requirements that denote dependencies	✘	✓	✘
3.4.7. Resolve deficiencies and conflicts between requirement sets	✘	✓	✘
3.4.8. Determine risk mitigations for each resource	✘	✓	✘
3.5. Execute Risk Management Framework	✘	✘	✓
3.6. Requirements for the operational environment			
3.6.1. Identify requirements and assumptions for hosts	✘	✓	✘
3.6.2. Identify requirements and assumptions for network	✘	✓	✘
3.7. Define use scenarios for threat modeling	✓	✘	✘
Architectural Design			
4.1. Risk assessment for 3rd party products			
4.1.1. Gather information about reputation of COTS	✘	✘	✓

Fig. A.2. Matrix: part II of V.

important observation in this context. Finally, a *research agenda* has been articulated by discussing a number of improvements that are relevant for all processes, including the more systematic support for verification and the optimization of processes for moving targets. While the overall setup of the paper is mainly based on a theoretical study of the processes, a small case study has been discussed in addition that confirms the theoretical results.

As ongoing research, the authors are working on combining the strong points of all approaches in order to distill an improved, consolidated process. A first activity will consist of selecting and combining the strongest practices for



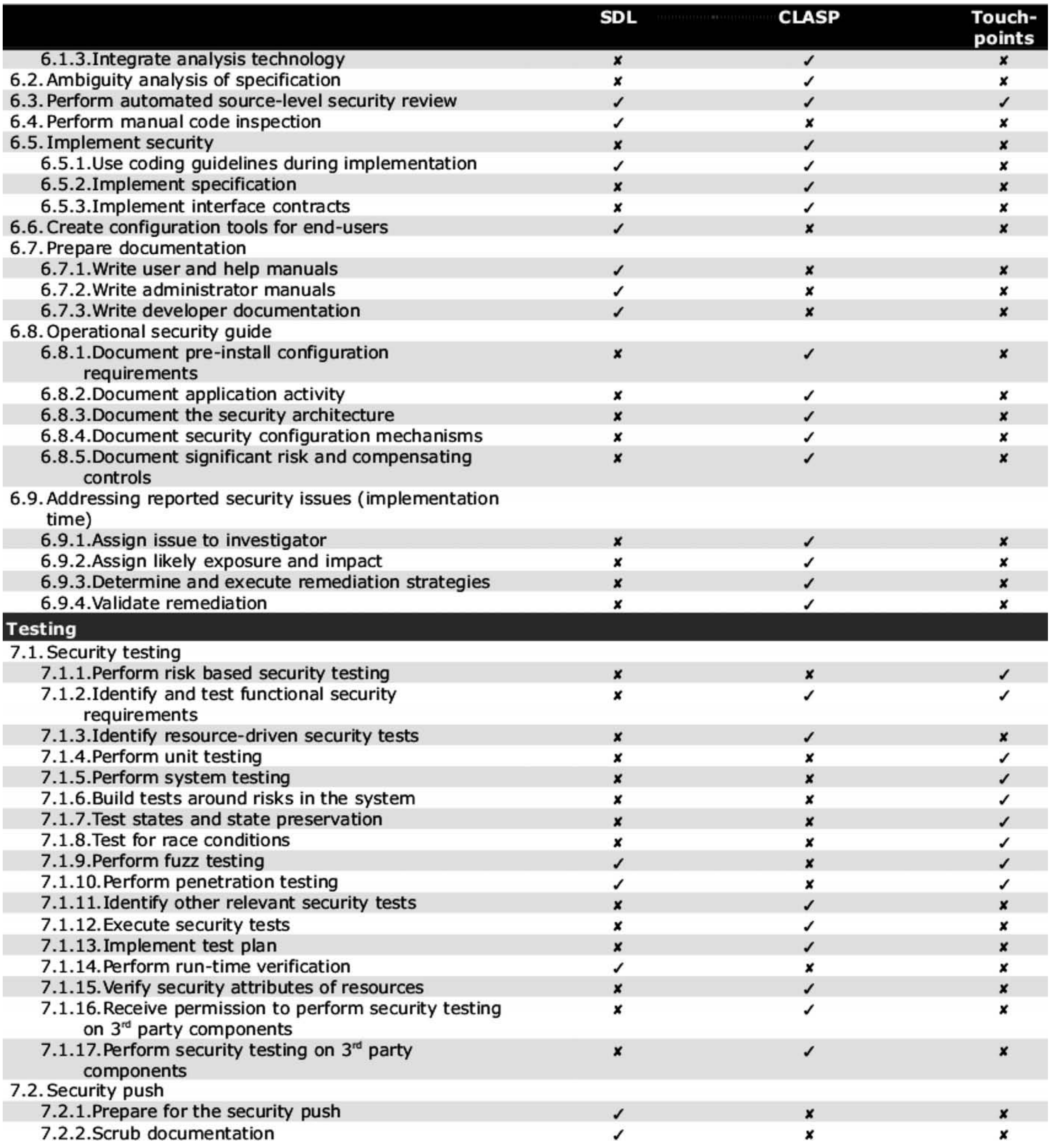
	SDL	CLASP	Touch-points
components and platforms			
4.1.2. Get technology assessment from vendor	✗	✓	✗
4.1.3. Perform security risk assessment for 3rd party technology	✓	✓	✗
4.2. Requirements audit			
4.2.1. Review non-security requirements	✗	✓	✗
4.2.2. Assess completeness of security requirements	✗	✓	✗
4.3. Architecture-level threat modeling			
4.3.1. Develop an understanding of the system	✗	✓	✓
4.3.2. Gather a list of external dependencies	✓	✗	✗
4.3.3. Define and validate security assumptions	✓	✓	✗
4.3.4. Create external security notes	✓	✗	✗
4.3.5. Create data flow diagrams	✓	✗	✗
4.3.6. Identify threat types	✓	✗	✓
4.3.7. Identify threats	✓	✓	✓
4.3.8. Perform ambiguity analysis	✗	✓	✓
4.3.9. Perform weakness analysis	✗	✗	✓
4.3.10. Assign risk to threats	✓	✓	✓
4.3.11. Identify countermeasures	✓	✓	✗
4.3.12. Create audit report and evaluate findings	✗	✓	✗
4.3.13. Review threat models	✓	✗	✗
4.3.14. Update threat model	✓	✗	✗
4.4. Security design principles			
4.4.1. Refine existing application security policy profile	✗	✓	✗
4.4.2. Determine implementation strategy for security services	✗	✓	✗
4.4.3. Build hardened protocol interfaces	✗	✓	✗
4.4.4. Design hardened application interfaces	✗	✓	✗
Detailed Design			
5.1. Assess the privacy impact rating of the project	✓	✗	✗
5.2. Software attack surface reduction			
5.2.1. Remove unimportant features	✓	✗	✗
5.2.2. Determine who needs access from where	✓	✗	✗
5.2.3. Reduce privileges	✓	✗	✗
5.2.4. Identify system entry points	✗	✓	✗
5.2.5. Map roles to entry points	✗	✓	✗
5.2.6. Map resources to entry points	✗	✓	✗
5.2.7. Scrub attack-surface	✓	✗	✗
5.3. Class design annotation			
5.3.1. Map data elements to resources and capabilities	✗	✓	✗
5.3.2. Annotate fields with policy information	✗	✓	✗
5.3.3. Annotate methods with policy data	✗	✓	✗
5.4. Database security configuration			
5.4.1. Identify candidate configuration	✗	✓	✗
5.4.2. Validate configuration	✗	✓	✗
5.5. Make your product updatable	✓	✗	✗
Implementation			
6.1. Security analysis tools for source management process			
6.1.1. Select analysis technology or technologies	✓	✓	✗
6.1.2. Determine analysis integration point	✗	✓	✗

Fig. A.3. Matrix: part III of V.

all important activities in the secure development life-cycle. Afterwards, complementary activities can then be selected in order to improve the overall coverage of the process. Finally, extra support for most of the areas of improvement that were discussed in the second part of the paper will have to be incorporated into the process. For some of these, like metrics, a small team effort will clearly not suffice to solve this challenging puzzle.

Appendix A. The activity-matrix

The matrix contains a section for each of the development phases, with each row representing a certain activity and each column representing one of the approaches. More information about the matrix, and in particular per-process links to the sources of the activities, can be found in a technical report [22]. See Figs. A.1–A.5.



	SDL	CLASP	Touch-points
6.1.3. Integrate analysis technology	✗	✓	✗
6.2. Ambiguity analysis of specification	✗	✓	✗
6.3. Perform automated source-level security review	✓	✓	✓
6.4. Perform manual code inspection	✓	✗	✗
6.5. Implement security	✗	✓	✗
6.5.1. Use coding guidelines during implementation	✓	✓	✗
6.5.2. Implement specification	✗	✓	✗
6.5.3. Implement interface contracts	✗	✓	✗
6.6. Create configuration tools for end-users	✓	✗	✗
6.7. Prepare documentation			
6.7.1. Write user and help manuals	✓	✗	✗
6.7.2. Write administrator manuals	✓	✗	✗
6.7.3. Write developer documentation	✓	✗	✗
6.8. Operational security guide			
6.8.1. Document pre-install configuration requirements	✗	✓	✗
6.8.2. Document application activity	✗	✓	✗
6.8.3. Document the security architecture	✗	✓	✗
6.8.4. Document security configuration mechanisms	✗	✓	✗
6.8.5. Document significant risk and compensating controls	✗	✓	✗
6.9. Addressing reported security issues (implementation time)			
6.9.1. Assign issue to investigator	✗	✓	✗
6.9.2. Assign likely exposure and impact	✗	✓	✗
6.9.3. Determine and execute remediation strategies	✗	✓	✗
6.9.4. Validate remediation	✗	✓	✗
Testing			
7.1. Security testing			
7.1.1. Perform risk based security testing	✗	✗	✓
7.1.2. Identify and test functional security requirements	✗	✓	✓
7.1.3. Identify resource-driven security tests	✗	✓	✗
7.1.4. Perform unit testing	✗	✗	✓
7.1.5. Perform system testing	✗	✗	✓
7.1.6. Build tests around risks in the system	✗	✗	✓
7.1.7. Test states and state preservation	✗	✗	✓
7.1.8. Test for race conditions	✗	✗	✓
7.1.9. Perform fuzz testing	✓	✗	✓
7.1.10. Perform penetration testing	✓	✗	✓
7.1.11. Identify other relevant security tests	✗	✓	✗
7.1.12. Execute security tests	✗	✓	✗
7.1.13. Implement test plan	✗	✓	✗
7.1.14. Perform run-time verification	✓	✗	✗
7.1.15. Verify security attributes of resources	✗	✓	✗
7.1.16. Receive permission to perform security testing on 3rd party components	✗	✓	✗
7.1.17. Perform security testing on 3rd party components	✗	✓	✗
7.2. Security push			
7.2.1. Prepare for the security push	✓	✗	✗
7.2.2. Scrub documentation	✓	✗	✗

Fig. A.4. Matrix: part IV of V.

	SDL	CLASP	Touch-points
7.3. Final security review			
7.3.1. Coordinate product Team	✓	✘	✘
7.3.2. Review unfixed security bugs	✓	✘	✘
7.3.3. Validate correct use of tools	✓	✘	✘
Release & Deployment			
8.1. Code sign-off	✓	✘	✘
8.2. Upload debugging symbols to central server	✓	✘	✘
8.3. Code signing			
8.3.1. Obtain code signing credentials	✘	✓	✘
8.3.2. Identify signing targets	✘	✓	✘
8.3.3. Sign identified targets	✘	✓	✘
8.4. Fine-tune access controls (network & OS) and configure the monitoring and logging	✘	✘	✓
Support			
9.1. Security response planning			
9.1.1. Build a security response center	✓	✘	✘
9.1.2. Provide means of communication for security issues	✘	✓	✘
9.1.3. Create your response team	✓	✘	✘
9.1.4. Find the vulnerabilities before researchers do	✓	✘	✘
9.2. Addressing deployment-time security issues (Security response execution)			
9.2.1. Decide on what to skip	✓	✘	✘
9.2.2. Vulnerability reporting & acknowledgment	✓	✓	✘
9.2.3. Determine vulnerability impact (triaging)	✓	✘	✘
9.2.4. Manage relationship with bug reporter	✓	✓	✘
9.2.5. Create security bulletin/advisory (Content creation)	✓	✘	✘
9.2.6. Create the fix	✓	✓	✘
9.2.7. Test the fix	✓	✘	✘
9.2.8. Release the fix and the bulletin/advisory	✓	✓	✘
9.2.9. Improve process based on lessons learned	✓	✘	✘

Fig. A.5. Matrix: part V of V.

References

[1] M. Howard, S. Lipner, The Security Development Lifecycle (SDL): A Process for Developing Demonstrably More Secure Software, Microsoft Press, 2006.

[2] OWASP, Comprehensive, lightweight application security process, http://www.owasp.org, 2006.

[3] G. McGraw, Software Security: Building Security, Addison Wesley, 2006.

[4] MSDN: Security development lifecycle phases, http://msdn2.microsoft.com/en-us/library/ms995349.aspx, 2005.

[5] Information technology security techniques evaluation criteria for it security, standard ISO/IEC 15408 (2005).

[6] B. Nuseibeh, Weaving together requirements and architectures, IEEE Computer 34 (3) (2001) 115–117. URL://citeseer.ist.psu.edu/nuseibeh01weaving.html .

[7] Microsoft threat modeling tool 2.1.2, http://www.microsoft.com/downloads/details.aspx?familyid=59888078-9daf- 4e96-b7d1-944703479451, 2007.

[8] T. Davenport, Process Innovation: Reengineering Work Through Information Technology, Harvard Business School Press, Boston, 1993.

[9] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

[10] K. Beck, Extreme Programming Explained, Addison-Wesley, 1999.

[11] Systems security engineering capability maturity model (SSE-CMM), standard ISO/IEC 21827, 2006.

[12] A. Jaquith, Security Metrics: Replacing Fear, Uncertainty, and Doubt, Addison-Wesley Professional, 2007.

[13] J.H. Saltzer, M.D. Schroeder, The protection of information in computer systems, Proceedings of the IEEE 63 (9) (1975) 1278–1308.

[14] J. Viega, G. McGraw, Building Secure Software: How to Avoid Security Problems the Right Way, Addison-Wesley, 2002.

[15] G. Stoneburner, C. Hayden, A. Feringa, Engineering principles for information technology security, NIST Special Publication 800-27, Revision A, 2004.

[16] Build security in, http://buildsecurityin.us-cert.gov/, 2007.

[17] G. Peterson, Collaboration in a secure development process, http://www.arctecgroup.net/articles.htm (June 2004).

[18] A. Hall, R. Chapman, Correctness by construction: developing a commercial secure system, IEEE Software 19 (1) (2002) 18–25. URL://citeseer.ist.psu.edu/hall02correctness.html .

[19] K. Goertzel, T. Winograd, H. McKinley, L. Oh, M. Colon, T. McGibbon, E. Fedchak, R. Vienneau, Software security assurance: A state-of-art report (soar), Tech. rep., Information Assurance Technology Analysis Center (IATAC), 2007.

[20] M. Howard, D.E. Leblanc, Writing Secure Code, Microsoft Press, Redmond, WA, USA, 2002.

[21] D. Wheeler, Secure programming for linux and unix howto, http://citeseer.ist.psu.edu/wheeler00secure.html.

[22] K. Buyens, J. Gregoire, B.D. Win, R. Scandariato, W. Joosen, Similarities and differences between clasp, sdl, and touchpoints: the activity-matrix, Tech. rep., K.U. Leuven, Department of Computer Science (October 2007).