ROYAUME DU MAROC
ADMINISTRATION DE LA DÉFENSE NATIONALE
DIRECTION GÉNÉRALE DE LA SÉCURITÉ
DES SYSTÈMES D'INFORMATION

المملكة المغربية
إدارة الدفاع الوطني
المديرية العامة لأمن
نظم المعلومات

Évaluation de la maturité de la sécurité du cycle de vie des développements logiciels

Guide de bonnes pratiques

2021

Évaluation de la maturité de la sécurité du cycle de vie des développements logiciels

Guide de bonnes pratiques

UNE PUBLICATION DE LA

DIRECTION GENERALE DE LA SECURITE DES SYSTEMES D'INFORMATION

2021

contact@dgssi.gov.ma

TABLE DES MATIERES

1 INTRODUCTION 5

2 METHODOLOGIES DE DEVELOPPEMENT LOGICIEL 7

2.1 AGILE 7
2.2 LEAN 8
2.3 CASCADE 8
2.4 ITERATIF 9
2.5 SPIRAL 9
2.6 DEVOPS 10
2.7 SYNTHESE 10

3 PRODUCTION MODERNE DE SERVICES IT 12

3.1 INTEGRATION CONTINUE CI - DEVOPS 12
3.2 DEPLOIEMENT CONTINUE CD 13
3.3 INTEGRATION DE LA SECURITE - DEVSECOPS 13
3.4 SYNTHESE 15

4 CYCLE DE VIE DE DEVELOPPEMENT SECURISE 16

4.1 MICROSOFT SECURITY DEVELOPMENT LIFECYCLE (MICROSOFT SDL) 16
4.2 COMPREHENSIVE LIGHTWEIGHT APPLICATION SECURITY PROCESS (CLASP) 17
4.3 ORACLE SOFTWARE SECURITY ASSURANCE (OSSA) 18
4.4 BUILDING SECURITY IN MATURITY MODEL (BSIMM) 19
4.5 TEAM SOFTWARE PROCESS – SECURE (TSP-SECURE) 20
4.6 SOFTWARE ASSURANCE MATURITY MODEL (SAMM) 22
4.7 SYNTHESE 23

5 MODELE DE MATURITE SAMM 26

5.1 NIVEAUX DE MATURITE 26
5.2 GOVERNANCE 27
5.3 DESIGN 29
5.4 IMPLEMENTATION 31
5.5 VERIFICATION 33
5.6 OPERATIONS 35
5.7 SYNTHESE 36

6 MATRICE DE CALCUL DU NIVEAU DE LA MATURITE 37

6.1 PREPARATION 37
6.2 EVALUATION 38
6.3 DEFINITION DE LA CIBLE SOUHAITEE 38
6.4 DEFINITION DU PLAN 38
6.5 MISE EN PLACE 39
6.6 MISE A DISPOSITION 39
6.7 EXEMPLE DE LA SIMULATION DE LA MATRICE 39
6.8 SYNTHESE 40

7 BONNES PRATIQUES SDLC 41

7.1 CORRECTION DES LOGICIELS ET DES SYSTEMES 41
7.2 FORMATION DES UTILISATEURS 41
7.3 AUTOMATISATION DES TACHES DE ROUTINE 41
7.4 APPLICATION DU MOINDRE PRIVILEGE 41
7.5 CREATION D'UN PLAN DE REPONSE AUX INCIDENTS SOLIDE 41
7.6 DOCUMENTATION DES POLITIQUES DE SECURITE 41
7.7 SEGMENTATION DU RESEAU 42
7.8 INTEGRATION DE LA SECURITE DANS LE CYCLE DE VIE DE DEVELOPPEMENT LOGICIEL 42
7.9 LOGGING ET MONITORING 42
7.10 DEFINITION DES INDICATEURS CLES 42
7.11 SENSIBILISATION, EDUCATION ET FORMATION 42
7.12 SYNTHESE 42

8 CADRE DE DEVELOPPEMENT SECURISE (MODELE NIST) 44

8.1 PREPARER L'ORGANISATION (PO) 45
8.2 PROTEGER LE LOGICIEL (PS) 46
8.3 PRODUIRE DES LOGICIELS BIEN SECURISES (PW) 46
8.4 REPONDRE AUX RAPPORTS DE VULNERABILITES (RV) 49
8.5 SYNTHESE 50

9 CONCLUSION 51

10 REFERENCES 52

11 LISTE DES FIGURES 53

12 LISTE DES TABLEAUX 54

13 ACRONYMES 55

1 Introduction

La sécurité informatique est un terme générique qui s'applique aux réseaux, à Internet, aux API, au cloud, aux applications, à la sécurité des conteneurs et autres. Elle consiste à établir un ensemble de stratégies de sécurité qui fonctionnent conjointement pour aider à protéger les données numériques. Il n'y a pas si longtemps, la sécurité informatique n'était contrôlée qu'à la fin d'un cycle de développement. Le processus était lent. Les entreprises recherchent actuellement des moyens pour créer un programme de sécurité intégré qu'elles seront en mesure d'adapter plus facilement et plus rapidement. La sécurité est donc intégrée dès la conception au lieu d'être ajoutée plus tard. Il faut veiller à intégrer au plus tôt la sécurité dans l'infrastructure et dans le cycle de vie des produits. Ainsi, elle sera à la fois proactive et réactive.

La sécurité continue repose sur un système régulier à base de feedbacks et des adaptations généralement gérés au moyen de points de contrôle automatisés. Grâce à l'automatisation, le feedback est rapide et efficace. Il ne ralentit pas le cycle de vie du produit. Cette méthode d'intégration de la sécurité permet de mettre en œuvre les mises à jour et les réponses aux incidents rapidement et globalement dans un environnement en constante évolution. Traditionnellement, la sécurité informatique consistait avant tout à renforcer, maintenir et contrôler le périmètre des datacenters, mais aujourd'hui ce périmètre tend à disparaître. Les méthodes de développement, de déploiement, d'intégration et de gestion informatiques sont en pleine mutation. Avec l'arrivée des clouds publics et hybrides, les responsabilités en matière de sécurité et de conformité réglementaire sont désormais partagées entre différents fournisseurs. L'adoption massive des conteneurs a fait surgir le besoin d'instaurer de nouvelles méthodes d'analyse, de protection et de mise à jour de la distribution des applications. Les applications mobiles fonctionnent sur une multitude d'appareils différents et l'infrastructure repose de plus en plus sur des logiciels, plutôt que sur du matériel. Résultat : les méthodes traditionnelles de gestion de la sécurité sont dépassées. Pour suivre le rythme de la transformation numérique, les programmes de sécurité doivent être adaptés afin que celle-ci soit continue, intégrée et flexible.

Il devient de plus en plus primordial d'adopter une démarche intégrant la sécurité dès la conception d'un logiciel et jusqu'à sa livraison et sa mise en production. Avec l'émergence des cyberattaques, il est aujourd'hui nécessaire de se prémunir contre le vol de données ou les intrusions. L'objectif d'une telle démarche est de pouvoir mettre en place des processus de tests automatisés, et ce à chaque étape du projet, et également au niveau de l'infrastructure.

Lorsque nous parlons du choix d'une méthodologie de développement logiciel, nous insistons sur le fait qu'il ne suffit pas de parler d'outils ou de processus, il est autant nécessaire d'impliquer tous les collaborateurs au projet de développement logiciel. En effet, l'idée est de leur faire comprendre que la sécurité doit être la priorité et que tout le monde en est responsable. C'est pourquoi il est primordial de mettre en place des processus visant à intégrer la sécurité à tous les niveaux.

L'objectif est donc de pouvoir anticiper les problèmes de sécurité afin de ne pas avoir à les gérer pendant le cycle de vie du projet ou lors de son fonctionnement en production.

Pour sécuriser les applications durant tout leur cycle de vie, le présent document commence par définir la sécurité des applications selon plusieurs axes. Il expose par la suite les méthodologies de développement logiciel les plus utilisées de nos jours et présente plusieurs approches sécurisées des cycles de vie des développements des logiciels pour présenter à la fin le modèle de maturité SAMM 2.0 qui permettra à chaque partie prenante d'accéder à un formulaire de questions/réponses simple et concis qui alimente de façon transparente la matrice de calcul de son niveau de maturité, et ce, quel que soit la méthodologie de développement logiciel adoptée. Cette matrice est disponible sur le site web de la DGSSI et est en langue française.

A la fin du document, un chapitre est dédié aux meilleures pratiques en matière de sécurité logicielle.

2 Méthodologies de développement logiciel

Le cycle de vie du développement logiciel SDLC[1] est le contrôle orthographique du monde du développement logiciel - il peut signaler les erreurs de création de logiciels avant qu'elles ne soient découvertes (à un coût beaucoup plus élevé). Le processus SDLC comprend plusieurs étapes distinctes, notamment la planification, l'analyse, la conception, la construction, les tests, le déploiement et la maintenance.

Ce chapitre a pour objectif de passer en revue une brève description des six méthodologies SDLC les plus courantes afin d'aider chaque structure ou organisation à choisir celle qui convient le mieux à ses projets et à son contexte :

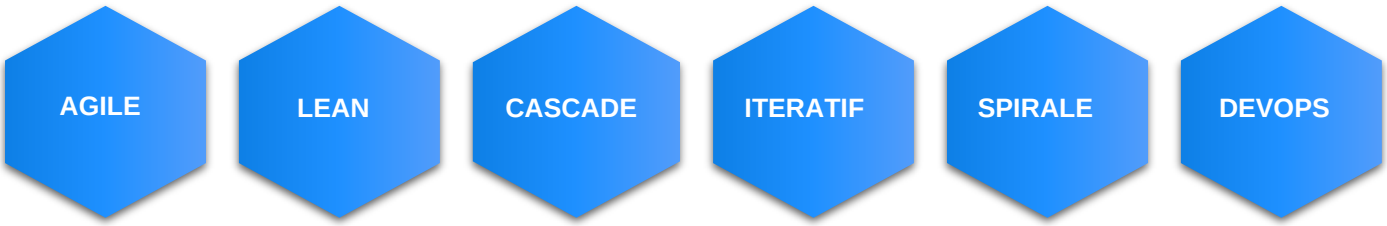


Figure 1 Méthodologies de développement logiciel

2.1 Agile

Le modèle Agile existe depuis environ une décennie. Mais ces derniers temps, il est devenu un moteur majeur du développement de logiciels dans de nombreuses organisations. Certaines entreprises valorisent tellement la méthodologie Agile qu'elles l'appliquent désormais à d'autres types de projets, y compris des initiatives non technologiques.

Dans le modèle Agile, "l'échec rapide" est une bonne chose. L'approche produit des cycles de versions en cours, chacune présentant de petits changements incrémentiels par rapport à la version précédente. A chaque itération, le produit est testé.

Le modèle Agile aide les équipes à identifier et à résoudre les petits problèmes sur les projets avant qu'ils ne se transforment en problèmes plus importants, à impliquer les parties prenantes de l'entreprise et à obtenir leurs commentaires tout au long du processus de développement.

Dans le cadre de leur adoption de cette méthodologie, de nombreuses équipes appliquent également un cadre Agile connu sous le nom de Scrum pour aider à structurer des projets de développement plus complexes.

Les équipes Scrum travaillent en "Sprints", qui durent généralement entre deux et quatre semaines, pour terminer les tâches assignées. Les réunions Scrum quotidiennes aident toute l'équipe à suivre les progrès tout au long du projet. Et le ScrumMaster est chargé de garder l'équipe concentrée sur son objectif.

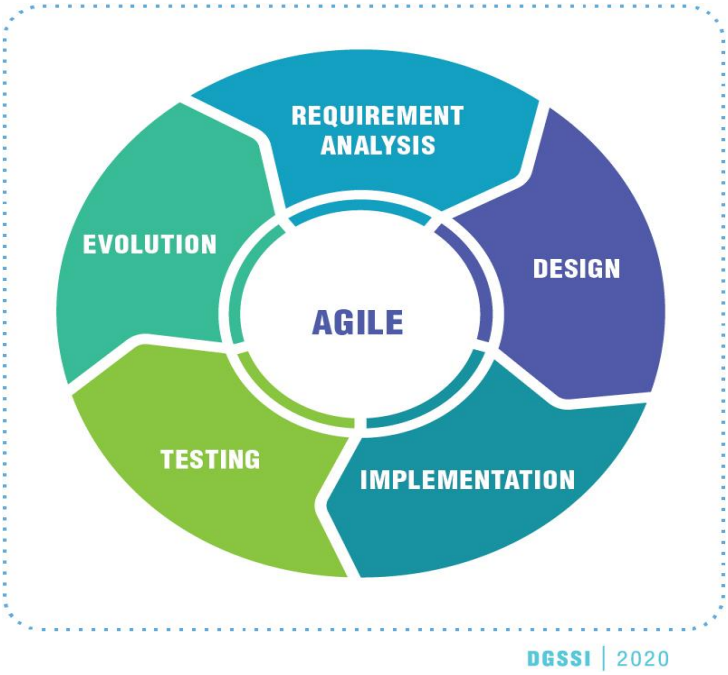


Figure 2 Méthodologie Agile

[1] SDLC: Software Development Life Cycle

2.2 Lean

Le modèle Lean pour le développement des logiciels est inspiré des pratiques et des principes de fabrication Lean. Les sept principes Lean (dans cet ordre) sont les suivants :

1. Éliminer le gaspillage ;
2. Amplifier l'apprentissage ;
3. Décider le plus tard possible ;
4. Livrer le plus rapidement possible ;
5. Responsabiliser l'équipe ;
6. Renforcer l'intégrité ;
7. Et voir l'ensemble.

Le processus Lean consiste à travailler uniquement sur ce qui doit être travaillé à ce moment-là, il n'y a donc pas de place pour le multitâche. Les équipes de projet se concentrent également sur la recherche d'opportunités pour réduire les déchets à chaque étape du processus SDLC, de l'abandon des réunions inutiles à la réduction de la documentation.

Figure 3 Principes Lean

Le modèle Agile est en fait, une méthode Lean pour le SDLC, mais avec quelques différences notables :

1. La première est la manière par laquelle chacun priorise la satisfaction du client : Agile en fait la priorité absolue dès le départ, créant un processus flexible où les équipes de projet peuvent répondre rapidement aux commentaires des parties prenantes tout au long du SDLC ;
2. Le Lean, quant à lui, met l'accent sur l'élimination des déchets comme moyen de créer plus de valeur globale pour les clients - ce qui, à son tour, contribue à améliorer la satisfaction.

2.3 Cascade

Certains experts soutiennent que le modèle en cascade n'a jamais été conçu pour être un modèle de processus pour gérer des projets réels. Quoi qu'il en soit, le modèle en cascade est largement considéré comme la plus ancienne des méthodologies SDLC structurées. C'est aussi une approche très simple :

Terminer une phase, puis passer à la suivante.

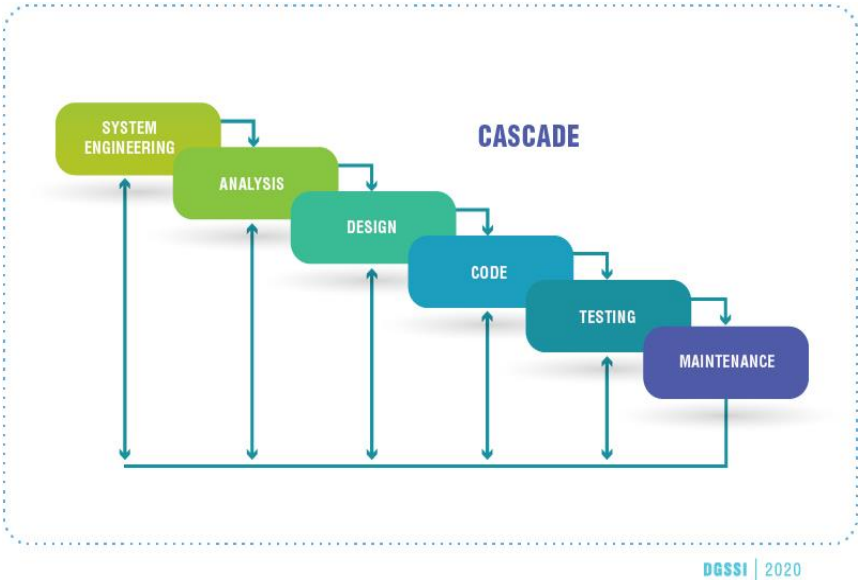


Figure 4 Modèle Cascade

Pas de retour. Chaque étape s'appuie sur les informations de l'étape précédente et possède son propre plan de projet.

L'inconvénient de la méthode en cascade est sa rigidité. Bien sûr, c'est facile à comprendre et simple à gérer. Mais les retards précoces peuvent perturber tout le calendrier du projet. Avec peu de place pour les révisions une fois qu'une étape est terminée, les problèmes ne peuvent être résolus qu'à l'étape de maintenance.

2.4 Itératif

Le modèle itératif est la répétition incarnée. Au lieu de commencer avec des exigences entièrement connues, les équipes de projet mettent en œuvre un ensemble d'exigences logicielles, puis testent, évaluent et identifient d'autres exigences. Une nouvelle version du logiciel est produite à chaque phase ou itération.

L'avantage du modèle itératif par rapport aux autres méthodologies SDLC courantes est qu'il produit une version fonctionnelle du projet au début du processus et rend la mise en œuvre des changements moins coûteuse.

L'inconvénient est que les processus répétitifs peuvent consommer des ressources rapidement.

Un exemple de modèle itératif est le RUP[2], développé par la division IBM Rational Software. RUP est, un produit de processus, conçu pour améliorer la productivité de l'équipe, qui capture également un bon nombre des meilleures pratiques dans le développement de logiciels modernes sous une forme adaptée à un large éventail de projets et d'organisations.

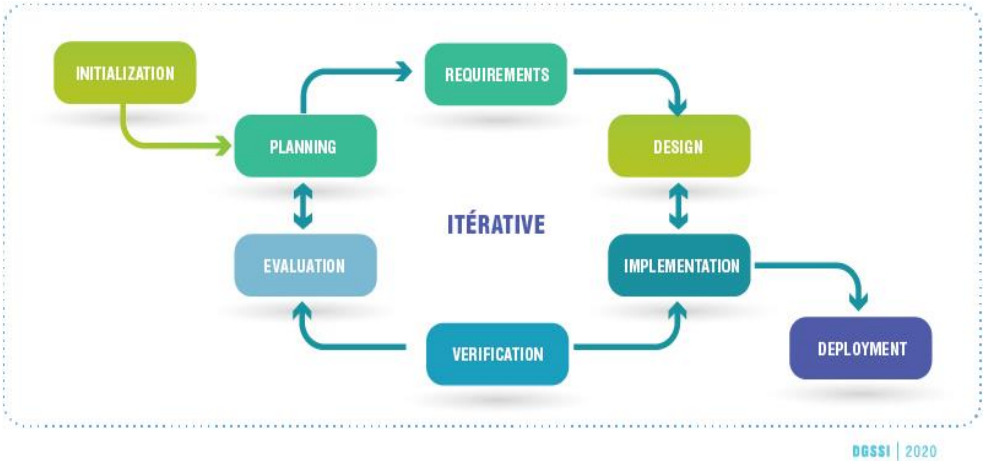


Figure 5 Modèle Itératif

RUP divise le processus de développement en quatre phases :

1. Le lancement, lorsque l'idée d'un projet est définie ;
2. L'élaboration, lorsque le projet est défini davantage et les ressources sont évaluées ;
3. La construction, lorsque le projet est développé et achevé ;
4. Et la transition, lorsque le produit est lancé.

Chaque phase du projet implique la modélisation, l'analyse et la conception, la mise en œuvre, les tests et enfin le déploiement.

2.5 Spiral

Le modèle spiral s'inspire du modèle itératif et de sa répétition ; le projet passe par quatre phases qui tournent à plusieurs reprises dans une spirale, jusqu'à ce que le SDLC soit terminé, ce qui permet de multiples cycles de raffinement :

1. La planification ;
2. L'analyse des risques ;
3. L'ingénierie ;
4. Et l'évaluation.

[2] RUP: Rational Unified Process

Le modèle Spiral est généralement utilisé pour les grands projets parce qu'il 'est l'une des méthodologies SDLC les plus flexibles.

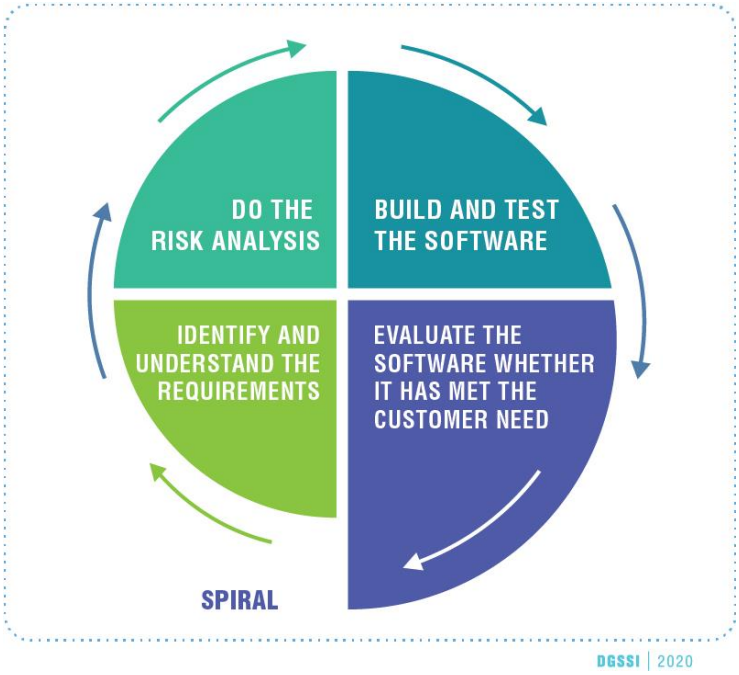


Figure 6 Modèle Spiral

Il permet aux équipes de développement de créer un produit hautement personnalisé et d'intégrer les commentaires des utilisateurs dès le début du projet. Un autre avantage de ce modèle SDLC est la gestion des risques. Chaque itération commence par anticiper les risques potentiels et déterminer la meilleure façon de les éviter ou de les atténuer.

2.6 DevOps

La méthodologie DevOps est le dernier arrivé sur la scène des méthodologies SDLC.
Elle s'inspire de deux tendances :

- **L'application des pratiques Agile et Lean au niveau de l'exploitation.**
- **La priorisation de la collaboration entre le développement et le personnel d'exploitation à toutes les étapes du processus SDLC.**

Dans un modèle DevOps, les développeurs et les équipes d'exploitation travaillent en étroite collaboration - et parfois en une seule équipe - pour accélérer l'innovation et le déploiement des produits logiciels et des fonctionnalités avec une meilleure qualité et avec plus de fiabilité. Les mises à jour des produits sont petites mais fréquentes. La discipline, la rétroaction continue et l'amélioration des processus, ainsi que l'automatisation des processus de développement manuels sont toutes des caractéristiques du modèle DevOps.

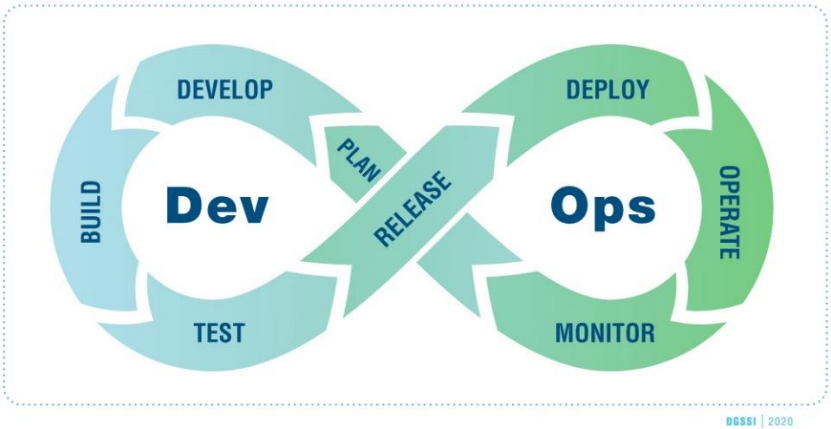


Figure 7 Modèle DevOps

2.7 Synthèse

Ces méthodologies de développement logiciel sont les plus répandues dans le développement logiciel. Chacune a ses propres forces et ses propres faiblesses et s'avère efficace dans différentes situations. Lors du choix de votre

méthodologie de développement, pensez à combiner les éléments de chaque méthode qui fonctionnent le mieux pour votre équipe et votre projet actuel. De cette manière, vous pouvez créer une méthodologie de développement hybride qui vous mènera à la production de manière sécurisée et efficace.

3 Production moderne de services IT

Avec les exigences de planning Time-to-market et le développement des méthodologies Agiles, les besoins d'industrialiser les tests, de fluidifier les déploiements en production et de rapprocher les équipes IT[3] se sont progressivement affirmés. Si les démarches Agiles veulent rapprocher les métiers des équipes de développement, le DevOps a pour vocation de rapprocher les fonctions de développement Dev[4] et d'exploitation Ops[5] de L'IT.

Les termes "Déploiement Continu" et "l'Intégration Continue" sont très présents depuis quelques années. Dans la méthodologie Agile, ces deux concepts s'intègrent de manière quasi-naturelle et bien souvent les équipes construisent leurs logiciels en suivant ces principes sans en avoir pleinement conscience.

3.1 Intégration continue CI - DevOps

L'intégration continue est un ensemble de pratiques consistant à tester de manière automatisée chaque révision de code avant de le déployer en production.

Lorsque le développeur code une fonctionnalité, il conçoit également le test associé et ajoute le tout à sa livraison de code. Le serveur d'intégration va ensuite exécuter tous les tests pour vérifier qu'aucune régression n'a été introduite dans le code source récemment livré. Si un problème est identifié, le déploiement n'a pas lieu et les Dev sont notifiés. Si aucune erreur n'est remontée, le serveur d'intégration peut déployer directement le code en production. Ainsi, avec l'intégration continue la phase de tests automatisés est complètement intégrée au flux de déploiement.

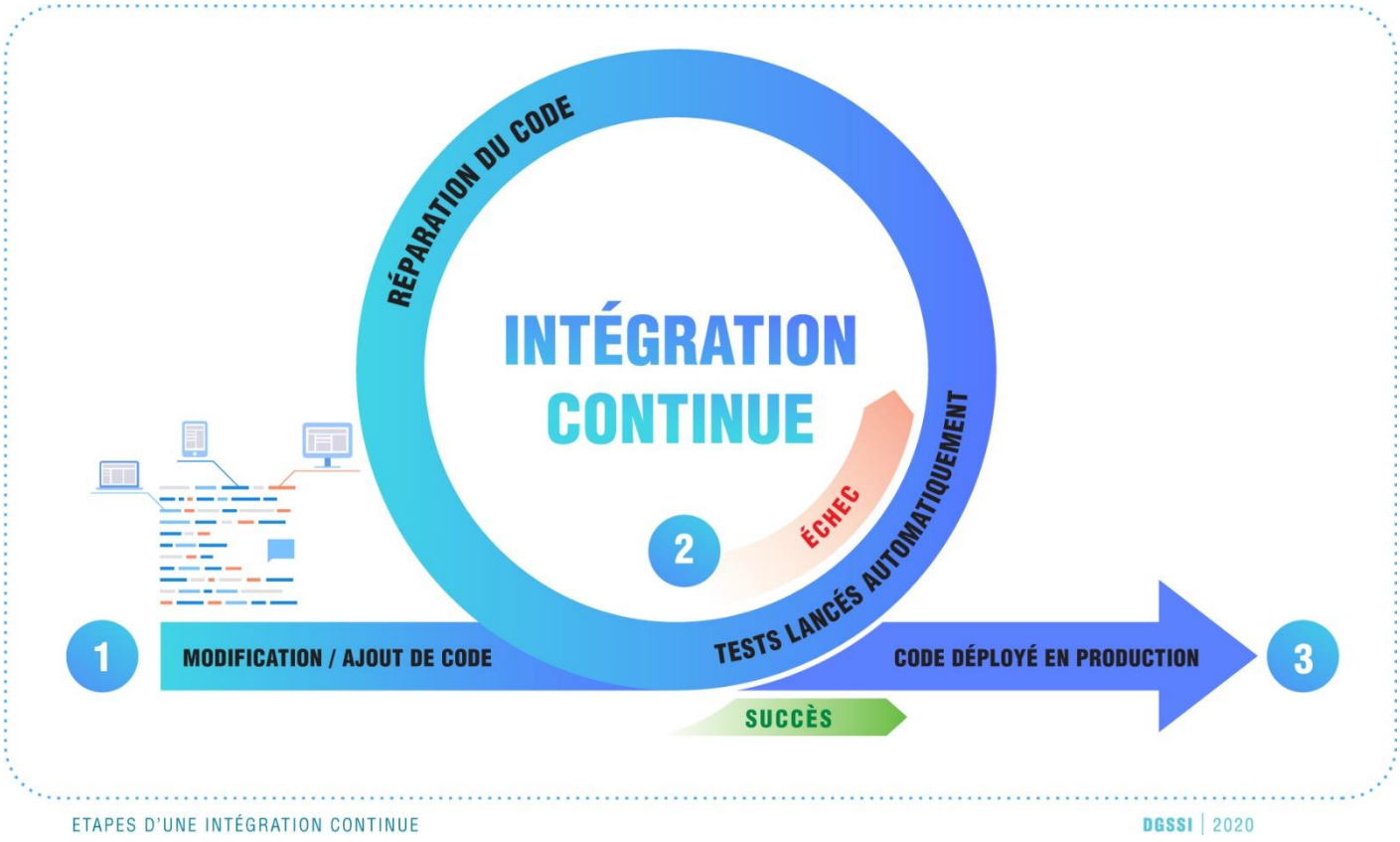


Figure 8 Etapes d'une Intégration Continue

L'intégration continue est comme un mur de qualité que le code doit franchir systématiquement avant d'être déployé en production. Cette pratique garantit un code de qualité, des utilisateurs finaux satisfaits et des développeurs plus efficaces. L'intégration continue CI[6] est un bon premier pas vers le déploiement continu CD[7].

Différents outils DevOps permettent de gérer l'intégration continue : Jenkins, Travis CI, Gitlab CI, …

[3] IT: Information Technology
[4] Dev: Development
[5] Ops: Operations
[6] CI: Continuous Integration
[7] CD: Continuous Delivery

3.2 Déploiement continue CD

L'intégration continue DevOps n'est pas un prérequis pour faire le déploiement continue, c'est plutôt un idéal à atteindre, un premier pas vers le déploiement continu CD. Les tests automatisés constituent un mur de qualité, certes non exhaustif, mais qui filtre la plupart des régressions s'il est bien conçu. Les Dev et les Ops sont ainsi plus sereins et plus aptes à déployer régulièrement.

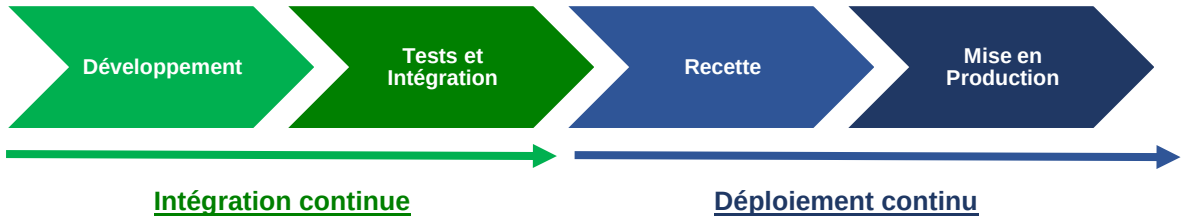


Figure 9 Chaine de production

Le déploiement continu est une pratique qui consiste à livrer chaque modification apportée au logiciel directement aux utilisateurs finaux. Aucune intervention humaine n'est nécessaire, tout se fait automatiquement. Seuls les changements qui échouent à un test ne sont pas directement déployés en production. Cette pratique permet d'accélérer la boucle de feedbacks, et permet aussi aux développeurs de se concentrer sur le développement de l'application puisqu'il n'y a plus de date de release à anticiper.

3.3 Intégration de la sécurité - DevSecOps

L'intégration continue CI et le déploiement continu CD sont de plus en plus répandus et utilisés. Le déploiement continu CD joue un rôle important dans les méthodes de développement Agile modernes en leur donnant des cycles de développement plus courts. DevOps, quant à lui, il prend en charge et met en œuvre le CD avec les bons outils.

De nos jours, les systèmes deviennent de plus en plus complexes et la sécurité n'est généralement pas intégrée au processus de développement. Elle est souvent prise en compte en dernier lieu. Le problème est que le CD comporte des versions continues pour lesquelles les méthodes de sécurité traditionnelles ne conviennent pas. Ces dernières années, un nouveau processus qui répartit les pratiques de sécurité entre le développement et l'exploitation est apparu sous le nom DevSecOps pour résoudre ce problème.

DevSecOps se concentre souvent sur une approche Agile du développement, qui cherche à conjuguer vitesse et efficacité. De plus en plus, les équipes travaillent ensemble de façon à traiter la sécurité au même niveau que le développement et l'exploitation. La production moderne de services IT est plus fiable, plus fluide et plus adaptative grâce à des processus itératifs et des outils d'automatisation.

De nombreux outils DevSecOps automatisent plusieurs tâches dans le cycle de vie du logiciel sans intervention humaine, (voir figure 10). D'autres outils DevSecOps, tels que des outils de collaboration et de communication, facilitent et stimulent l'interaction humaine pour améliorer la productivité. Certains outils interviennent dans une activité spécifique à une phase de cycle de vie.

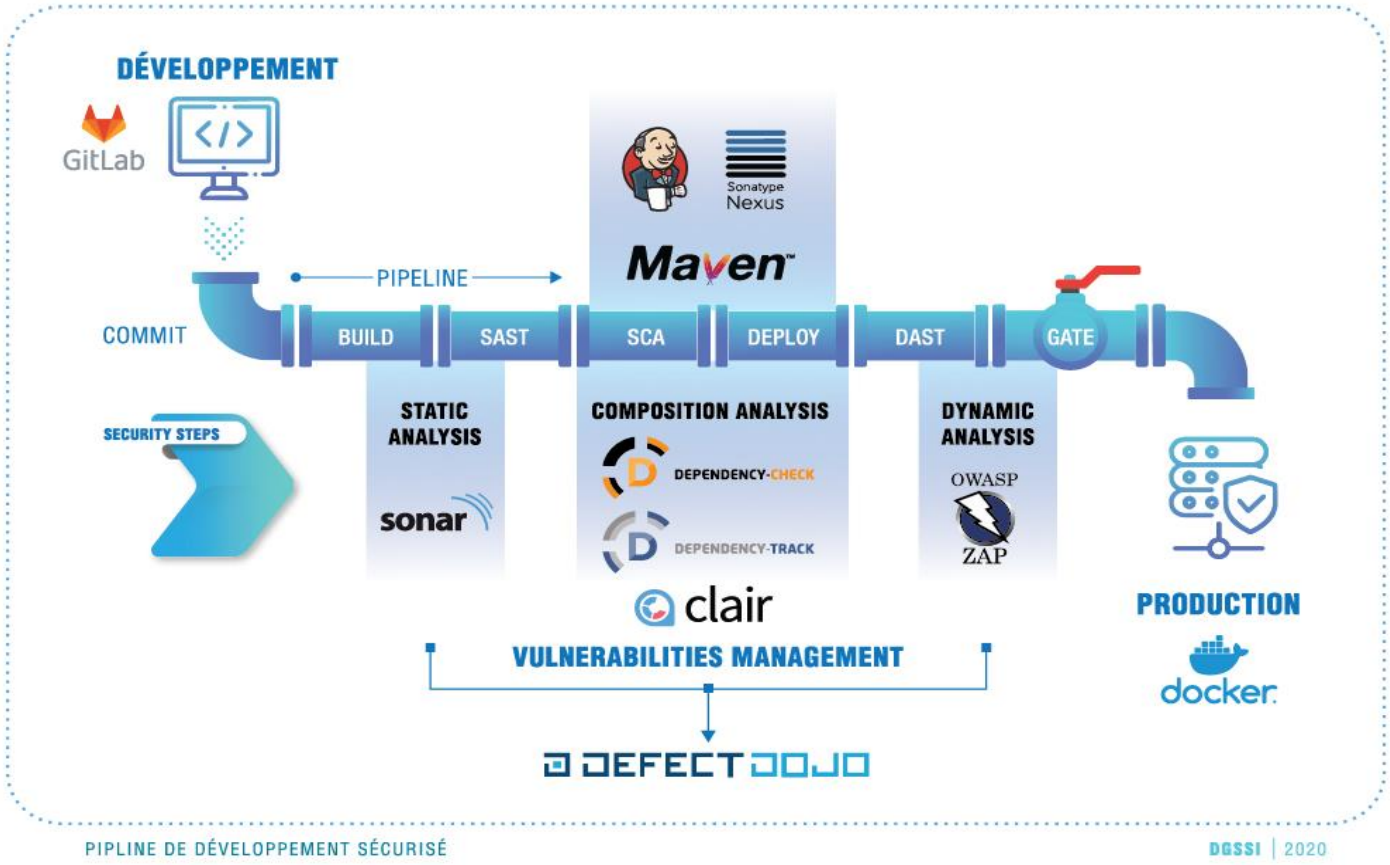


Figure 10 Cycle de vie d'une application logicielle moderne

3.3.1 Principes de base

DevSecOps est un mouvement qui est de plus en plus adopté par les équipes techniques au niveau des entreprises éditrices de logiciels et également au niveau des intégrateurs des solutions. Ce principe propose de mettre en place une jonction entre le groupe de développement, le groupe des métiers et aussi le groupe de gestion de la sécurité pour réaliser efficacement et de manière maîtrisée le produit final.

Il existe plusieurs principes clés pour mettre en œuvre cette nouvelle approche de façon réussie notamment :

- **Supprimer les goulots d'étranglement (y compris le facteur humain) et les actions manuelles ;**
- **Automatiser autant que possible les activités de développement et de déploiement ;**
- **Adopter des outils communs depuis la spécification des exigences jusqu'au déploiement et aux opérations ;**
- **Tirer parti des principes logiciels Agiles et privilégier les petites mises à jour incrémentielles et fréquentes par rapport aux versions plus importantes et plus sporadiques ;**
- **Appliquer les compétences transversales de développement, de sécurité et d'opérations tout au long du cycle de vie de l'application, en adoptant une approche de surveillance continue en parallèle au lieu d'attendre d'appliquer chaque compétence de manière séquentielle ;**
- **Mesurer et quantifier les risques de la sécurité de l'infrastructure sous-jacente, de manière à comprendre tous les risques et leurs impacts sur les applications ;**
- **Déployer une infrastructure immuable, telle que les conteneurs. Le concept d'infrastructure immuable est une stratégie informatique dans laquelle les composants déployés sont remplacés dans leur intégralité, plutôt que d'être mis à jour sur place. Le déploiement d'une infrastructure immuable nécessite la standardisation et l'émulation des composants des infrastructures communes pour obtenir des résultats cohérents et prévisibles.**

3.3.2 Phases du cycle de vie DevSecOps

Les phases du cycle de vie d'un logiciel selon une approche DevSecOps sont illustrées à la figure 11. Il y existe neuf phases : **planifier, développer, construire, tester, publier, livrer, déployer, exploiter et surveiller**. La sécurité est intégrée dans chacune de ces phases.

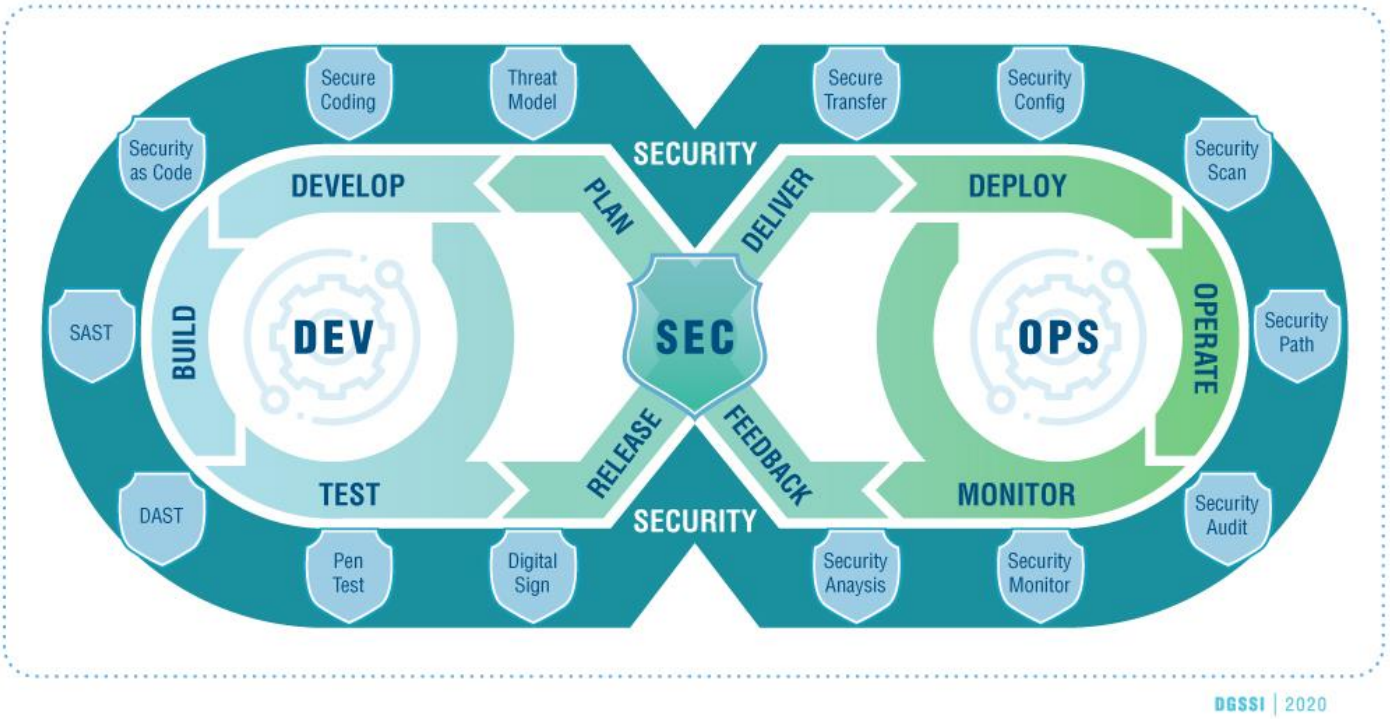


Figure 11 Cycle de vie d'un logiciel selon une approche DevSecOps

Avec la DevSecOps, le cycle de vie du développement logiciel n'est pas un processus linéaire monolithique. La livraison de style big bang dans le modèle cascade est remplacée par des livraisons petites mais plus fréquentes, de sorte qu'il est plus facile de changer de cap si nécessaire.

Chaque petite livraison est accomplie grâce à un processus entièrement automatisé ou semi-automatisé avec une intervention humaine minimale pour accélérer l'intégration et le déploiement continus. Le cycle de vie DevSecOps est adaptable et possède de nombreuses boucles de rétroaction pour une amélioration continue.

3.4 Synthèse

Le DevSecOps est le processus d'intégration des meilleures pratiques et méthodologies de développement sécurisé dans les processus de développement et de déploiement logiciel.

Plusieurs pratiques standards telles que les tests automatisés, l'intégration continue (CI) et le déploiement continu (CD) sont originaires du monde Agile, qui date (de manière informelle) des années 1990 et formellement de 2001. DevSecOps est une réponse naturelle et nécessaire pour combler les écarts traditionnels entre l'informatique et la sécurité tout en garantissant une livraison rapide et sûre du code. DevSecOps résout les problèmes de vitesse, de risque, de sécurité et de qualité logicielle. Il permet l'identification et la correction précoces des vulnérabilités du code. Il permet de rendre chacun responsable de la sécurité dans le but de mettre en œuvre les décisions et les actions relatives à la sécurité à la même échelle et à la même vitesse que les décisions et les actions liées aux développements et à l'exploitation.

4 Cycle de vie de développement sécurisé

Dans ce chapitre, plusieurs Frameworks intégrant la sécurité par défaut sont étudiés :

- **Cycle de vie du développement de la sécurité (Microsoft SDL)**
- **Processus complet de sécurité des applications légères (CLASP)**
- **Oracle Software Security Assurance (OSSA)**
- **Renforcement du cadre du modèle de sécurité dans la maturité (BSIMM)**
- **Processus logiciel d'équipe sécurisé (TSP-Secure)**
- **Modèle de maturité de la Software Assurance (SAMM)**

Ces méthodologies prennent en compte les activités liées à la sécurité couvrant l'ensemble du processus de développement logiciel.

4.1 Microsoft Security development lifecycle (Microsoft SDL)

Dans sa forme la plus simple, le SDL[8] est un processus qui standardise les meilleures pratiques de la sécurité à travers une gamme de produits et / ou d'applications. Il capture les activités de sécurité standards de l'industrie et les empaquette afin qu'elles puissent être facilement mises en œuvre. C'est un processus avec différentes phases qui contiennent des activités de sécurité situées à l'intérieur du triangle classique des personnes-processus-technologie : Le SDL constitue la partie "Processus".

La modélisation des menaces est un élément central du cycle de vie du développement de sécurité Microsoft SDL. Il s'agit d'une technique d'ingénierie qui sert à identifier les menaces, les attaques, les vulnérabilités et les contre-mesures qui pourraient affecter une application.

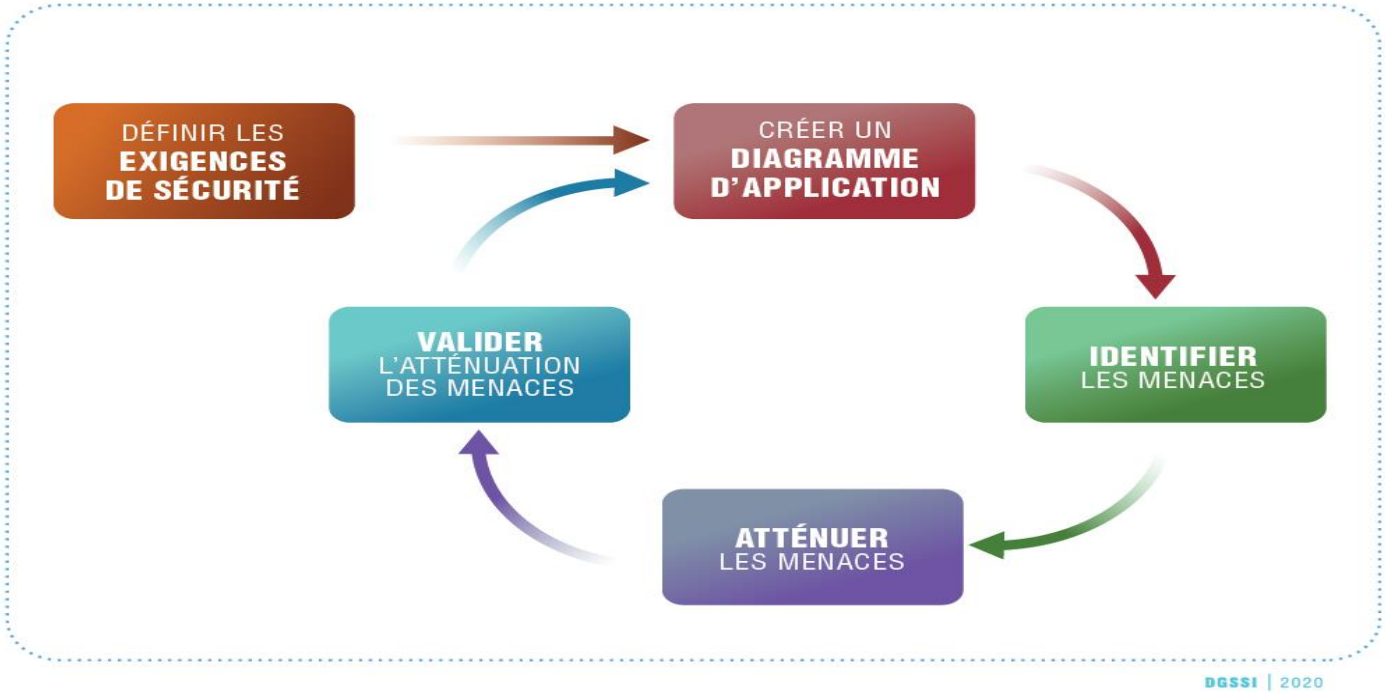


Figure 12 Les cinq étapes du modèle SDL

Le modèle SDL prévoit cinq étapes principales de modélisation des menaces :

1. **Définir les exigences de la sécurité ;**
2. **Créer un diagramme d'application ;**
3. **Identifier les menaces ;**
4. **Atténuer les menaces ;**
5. **Valider que les menaces ont été atténuées.**

[8] SDL: Software Development Lifecycle

4.2 Comprehensive Lightweight Application Security Process (CLASP)

Le processus de sécurité des applications complet et léger CLASP[9] est un ensemble de composants de processus axé sur l'activité, basé sur les rôles et guidé par les meilleures pratiques formalisées. Il est conçu pour aider les équipes de développement de logiciels à intégrer la sécurité dans les premières étapes des cycles de vie de développement de logiciels d'une manière structurée, reproductible et mesurable.

Le processus CLASP définit les rôles qui peuvent avoir un impact sur l'état de la sécurité d'un produit logiciel et attribue des activités à ces rôles. En tant que tels, les rôles sont utilisés comme une perspective supplémentaire pour structurer l'ensemble des activités. Les rôles sont responsables de la finalisation et de la qualité des résultats d'une activité.

La figure 13 présente le processus CLASP à travers cinq perspectives de haut niveau appelées Vues CLASP. Ces vues permettent une meilleure compréhension du processus CLASP et des interactions entre ses différents composants afin de faciliter leur application dans le cycle de vie de développement d'un logiciel spécifique.

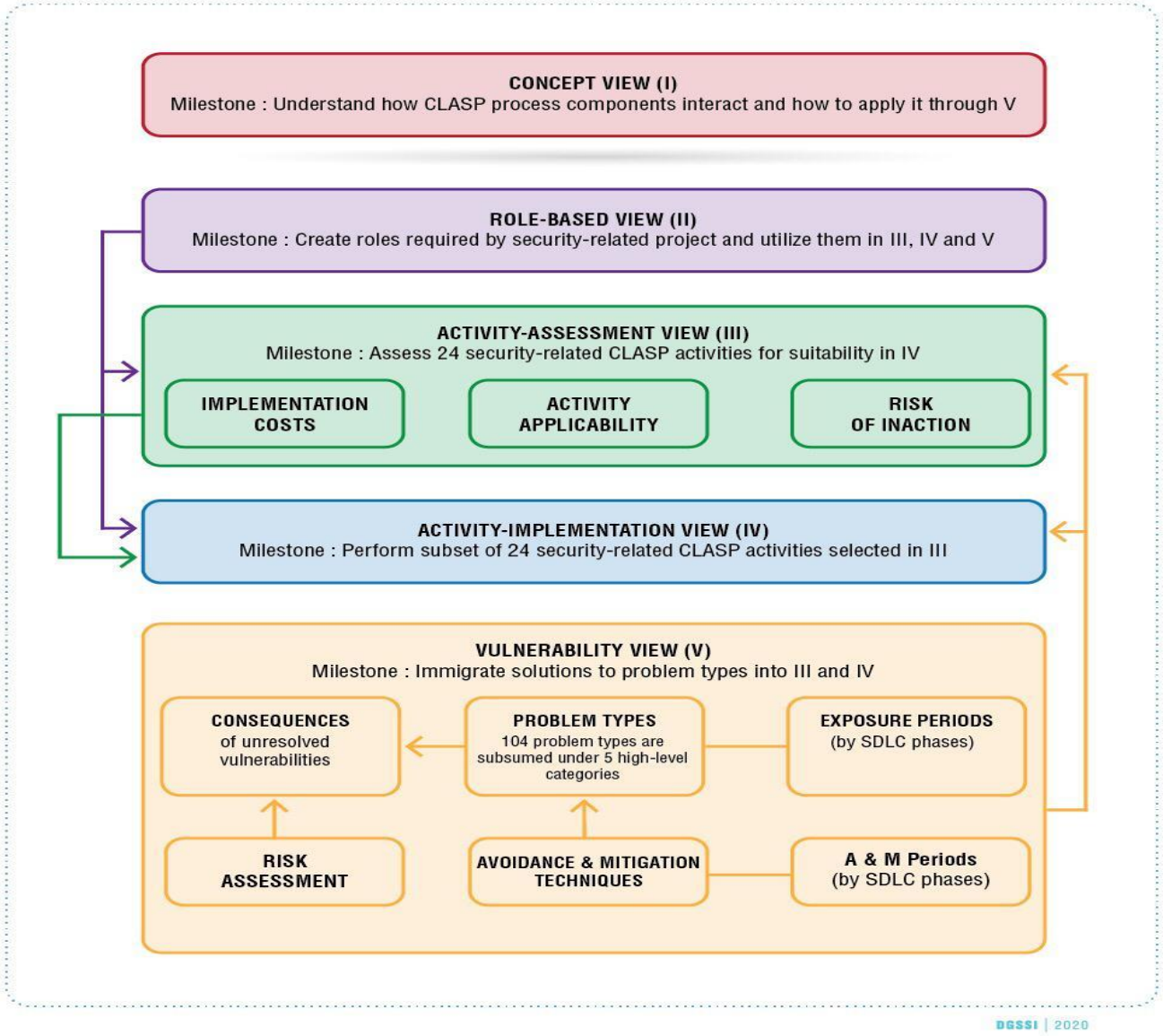


Figure 13 Processus CLASP

CLASP est défini comme un ensemble d'activités indépendantes qui doivent être intégrées dans le processus de développement et son environnement opérationnel. Le choix des activités à exécuter et l'ordre d'exécution sont laissés ouverts dans un souci de flexibilité. De plus, la fréquence d'exécution des activités est spécifiée par activité individuelle, ce qui entraîne une coordination complexe.

[9] CLASP: Comprehensive Lightweight Application Security Process

4.3 Oracle Software Security Assurance (OSSA)

Oracle Software Security Assurance est un ensemble de normes, de technologies et de pratiques strictes du secteur visant à :

- **Favoriser les innovations en matière de sécurité**
- **Réduire l'impact des vulnérabilités de tous les produits Oracle.**
- **Réduire l'impact des vulnérabilités des produits commercialisés sur les clients.**

4.3.1 Sécurité intégrée

Comprenant toutes les phases du cycle de vie du développement de produits, l'OSSA[10] est la méthodologie d'Oracle pour intégrer la sécurité dans la conception, la construction, les tests, la livraison et la maintenance de ses produits. L'objectif est de s'assurer que les produits d'Oracle, ainsi que les systèmes client qui exploitent ces produits, restent aussi sécurisés que possible.

4.3.2 Normes de codage sécurisé

Un logiciel sécurisé ne se produit pas tout seul. Cela nécessite des méthodologies appliquées de manière cohérente dans chaque organisation ; méthodologies conformes aux politiques, objectifs et principes énoncés. Oracle Secure Coding Standards est un guide pour les développeurs dans leurs efforts pour produire du code sécurisé. Les normes traitent de domaines de connaissances générales en matière de sécurité tels que les principes de conception et les vulnérabilités courantes, et fournissent des conseils spécifiques sur des sujets tels que la validation des données, la confidentialité des données, la gestion des utilisateurs, etc. Les normes de codage ont été développées au fil de nombreuses années et incorporent les meilleures pratiques ainsi que les leçons tirées des tests de vulnérabilité continus par l'équipe interne de piratage éthique d'Oracle. Les normes de codage sécurisé sont un élément clé de l'OSSA et le respect des normes est évalué et validé pendant toute la durée de vie utile de tous les produits Oracle.

4.3.3 Analyse et tests de sécurité complets

Les tests de sécurité chez Oracle comprennent des activités fonctionnelles et non fonctionnelles pour la vérification des caractéristiques et de la qualité du produit. Bien que ces types de tests ciblent souvent des fonctionnalités de produit qui se chevauchent, ils ont des objectifs différents et sont donc réalisés par différentes équipes. Les tests de sécurité fonctionnels et non fonctionnels se complètent pour fournir une couverture de sécurité complète. Les tests de sécurité fonctionnelle sont généralement exécutés par des équipes régulières d'assurance qualité des produits dans le cadre du cycle de test normal des produits. Au cours de ces tests, les ingénieurs QA[11] vérifient la conformité des fonctionnalités de sécurité implémentées à ce qui avait été préalablement convenu dans les spécifications fonctionnelles, lors du processus de révision de l'architecture et de la liste de contrôle. L'analyse et les tests de sécurité non fonctionnels vérifient l'assurance de sécurité des produits, y compris l'identification et la correction des vulnérabilités et la résistance aux attaques. Il existe deux grandes catégories de tests utilisés pour tester les produits : l'analyse statique et l'analyse dynamique. Ces tests s'intègrent différemment dans le cycle de vie du développement du produit et ont tendance à trouver différentes catégories de problèmes, ils sont donc utilisés ensemble par les équipes de sécurité.

4.3.4 Gestion et correction des vulnérabilités de sécurité

Le programme Critical Patch Update est le principal mécanisme de publication des correctifs de bogues de sécurité pour les produits Oracle. Les mises à jour critiques des correctifs sont publiées tous les trimestres, le mardi le plus proche du 17 du mois en janvier, avril, juillet et octobre. De plus, Oracle conserve la possibilité d'émettre des correctifs ou des instructions de contournement hors calendrier en cas de vulnérabilités particulièrement critiques et / ou lorsque des exploits actifs sont signalés dans la nature. Ce programme est connu sous le nom de programme d'alerte de sécurité.

[10] OSSA: Oracle Software Security Assurance
[11] QA: Quality Assurance

4.4 Building Security in Maturity Model (BSIMM)

Le modèle BSIMM[12] est un modèle basé sur les données pour aider les parties prenantes à comprendre, mesurer et planifier une initiative de sécurité logicielle. Il a été créé en observant et en analysant des données du monde réel issues d'initiatives de sécurité logicielle de premier plan.

Le BSIMM n'est pas une norme, c'est plutôt une description de l'ensemble des activités pratiquées par les initiatives de sécurité logicielle les plus réussies au monde. Il ne décrit pas ce qu'il faut faire pour la sécurisation des logiciels ; au lieu de cela, il indique ce que plusieurs entreprises font réellement.

La principale fonction d'organisation du BSIMM est son cadre de sécurité logicielle. Ce cadre comprend quatre domaines (Gouvernance, Intelligence, Points de contact SSDL, Déploiement) qui contiennent douze pratiques :

- Gouvernance :
  - Pratique 1 - Stratégie et métriques
  - Pratique 2 - Conformité et politique
  - Pratique 3 - Formation
- Intelligence :
  - Pratique 4 - Modèles d'attaque
  - Pratique 5 - Caractéristiques et conception de sécurité
  - Pratique 6 - Normes et exigences
- Points de contact SSDL :
  - Pratique 7 - Analyse d'architecture
  - Pratique 8 - Révision de code
  - Pratique 9 - Tests de sécurité
- Déploiement :
  - Pratique 10 - Tests de pénétration
  - Pratique 11 - Environnement logiciel
  - Pratique 12 - Gestion de la configuration et gestion des vulnérabilités

Gouvernance	Intelligence	Points de contact SSDL	Déploiement
Stratégie et métriques	Modèles d'attaque	Analyse d'architecture	Tests de pénétration
Conformité et politique	Caractéristiques et conception de sécurité	Révision de code	Environnement logiciel
Formation	Normes et exigences	Tests de sécurité	Gestion de la configuration et des vulnérabilités

Tableau 1 Pratiques BSIMM

La version 10 du BSIMM représente la dernière évolution de ce modèle. En moyenne, les 122 entreprises dont les données sont incluses dans BSIMM version 10 avaient pratiqué la sécurité logicielle pendant 4,53 ans au moment de leur évaluation actuelle (avec des valeurs allant de moins d'un an à 19 ans en juin 2019). Les 122 entreprises conviennent que le succès de leurs initiatives dépend de la création d'un groupe interne dédié à la sécurité des logiciels.

La taille moyenne du SSG[13] est de 13,1 personnes (la plus petite 1, la plus grande 160, la médiane de 6,0), avec un groupe satellite moyen d'autres personnes (développeurs, architectes et membres de l'organisation directement engagés et promouvant la sécurité logicielle) composé de 51,6 personnes (la plus petite 0, plus grand 1500, médiane 0). Le nombre moyen de développeurs parmi les participants était de 3 804,2 personnes (les 5 plus petits, 100 000 plus grands, la médiane 900), ce qui donne un pourcentage moyen de SSG au développement de 1,37% (médiane de 0,61%).

Au total, le BSIMM décrit le travail de 1 596 membres du SSG travaillant avec un satellite de 6 298 personnes pour sécuriser le logiciel - près de 175 000 applications - développé par 468 500 développeurs.

[12] BSIMM: Building Security in Maturity Model
[13] SSG: Software Systems Group

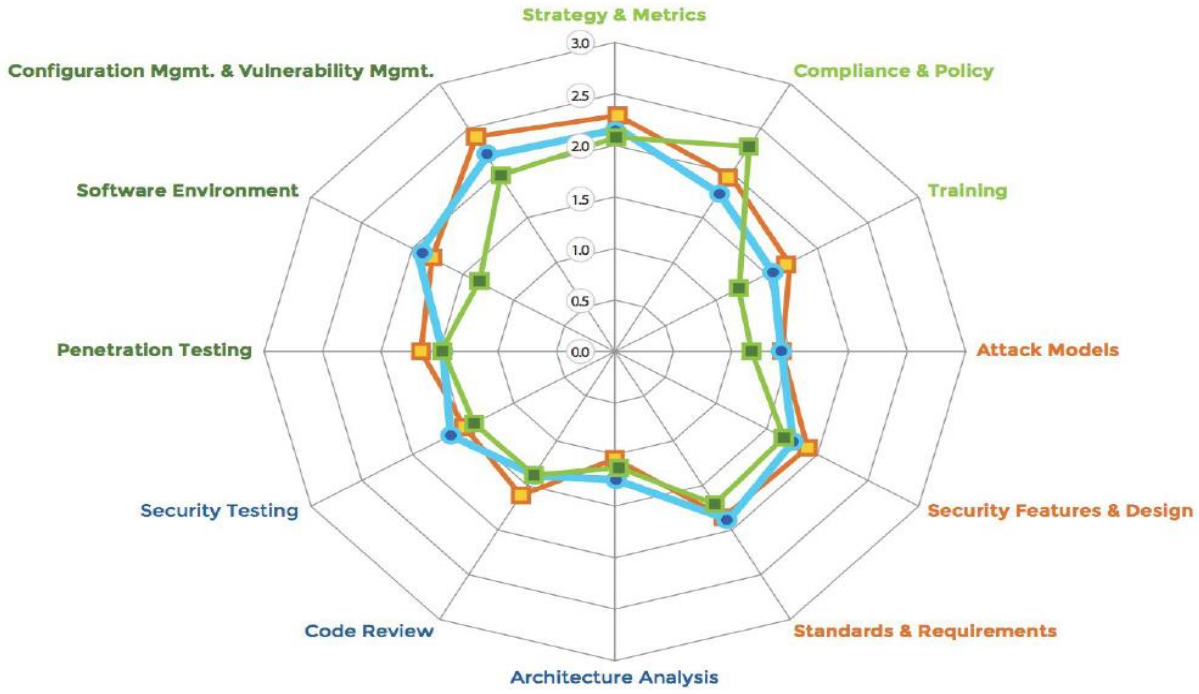


Figure 14 Activités de sécurité BSIMM

4.5 Team Software Process – Secure (TSP-Secure)

TSP-Secure est une méthode basée sur le TSP[14] (Processus Logiciel d'équipe) du Software Engineering Institute SEI[15] fournissant un cadre, un ensemble de processus et des méthodes pour gérer la sécurité au niveau du cycle de vie des développements logiciels.

TSP-Secure aborde le développement de logiciels sécurisés de trois manières. Premièrement, comme les logiciels sécurisés ne sont pas créés par accident, TSP-Secure gère la planification de la sécurité. De plus, étant donné que les contraintes d'horaire et les problèmes de personnel entravent la mise en œuvre des meilleures pratiques, TSP-Secure aide à constituer des équipes de développement autogérées, puis à les mettre en charge de leur propre travail. Deuxièmement, étant donné que la sécurité et la qualité sont étroitement liées, TSP-Secure aide à gérer la qualité tout au long du cycle de vie du développement du produit. Finalement, les équipes utilisant TSP-Secure élaborent leurs propres plans. La planification initiale se déroule dans le cadre d'une série de réunions appelées lancement de projet, qui se déroule sur une période de trois à quatre jours. Le lancement est dirigé par un coach d'équipe qualifié. Lors d'un lancement TSP-Secure, l'équipe parvient à une compréhension commune des objectifs de sécurité du travail et de l'approche qu'elle adoptera pour effectuer le travail, produit un plan détaillé pour guider le travail et obtient un soutien de la direction pour le plan. Les tâches typiques incluses dans le plan sont l'identification des risques de sécurité, l'obtention et la définition des exigences de sécurité, la conception sécurisée, la conception sécurisée et les revues de code, et l'utilisation d'outils d'analyse statique, de tests unitaires et de tests fuzz.

Chaque membre d'une équipe TSP-Secure sélectionne au moins l'un des neuf rôles de membre d'équipe standard (les rôles peuvent être partagés). L'un des rôles définis est un rôle Security Manager. Le responsable de la sécurité dirige l'équipe pour s'assurer que les exigences du produit, la conception, la mise en œuvre, les examens et les tests concernent la sécurité ; s'assurer que le produit est assuré statiquement et dynamiquement ; fournir des analyses et des alertes en temps opportun sur les problèmes de sécurité ; et le suivi de tous les risques ou problèmes de sécurité jusqu'à la fermeture. Le responsable de la sécurité travaille avec des experts en sécurité externes en cas de besoin.

Après le lancement, l'équipe exécute son plan et s'assure que toutes les activités liées à la sécurité ont lieu. L'état de la sécurité est présenté et discuté lors de chaque briefing sur l'état de la gestion.

La stratégie de gestion de la qualité de TSP-Secure consiste à avoir plusieurs points d'élimination des défauts dans le cycle de vie du développement logiciel. Plus il y a de points de suppression de défauts, plus il y a de chances de trouver des problèmes juste après leur introduction, ce qui permet de résoudre plus facilement les problèmes et de déterminer et de traiter plus facilement la cause première.

[14] TSP: Team Software Process
[15] SEI: Software Engineering Institute

Chaque activité de suppression de défaut peut être considérée comme un filtre qui supprime un certain pourcentage de défauts pouvant entraîner des vulnérabilités du produit logiciel (voir Figure 15). Plus il y a de filtres de suppression de défauts dans le cycle de vie du développement logiciel, moins il y aura de défauts pouvant entraîner des vulnérabilités dans le logiciel lors de sa sortie. Plus important encore, la mesure précoce des défauts permet à l'organisation de prendre des mesures correctives tôt dans le cycle de vie du développement logiciel.

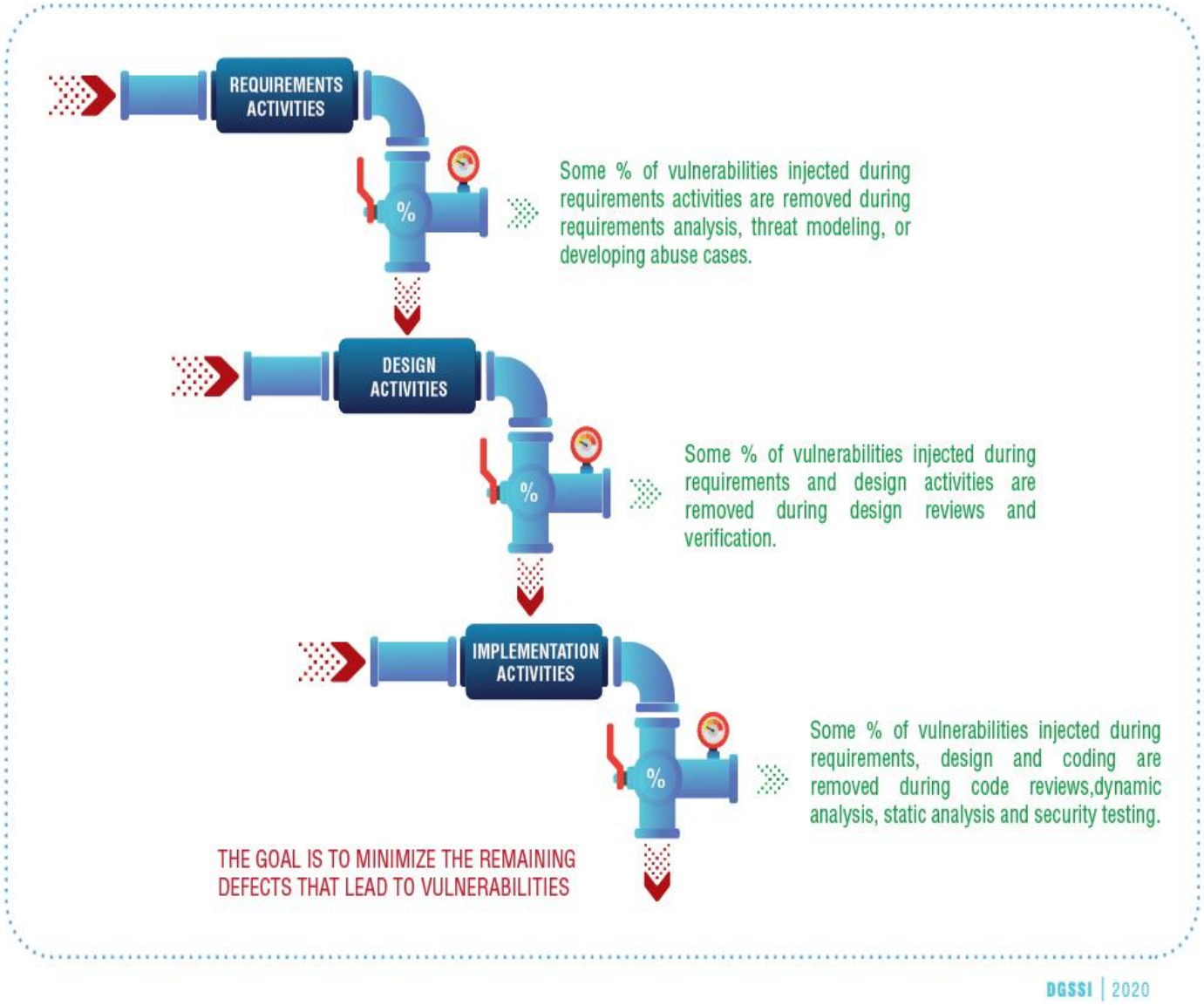


Figure 15 Filtres de suppression des vulnérabilités

4.6 Software Assurance Maturity Model (SAMM)

Le SAMM[16] est un modèle de maturité de l'OWASP[17] mis en place afin d'aider de manière efficace les entreprises désireuses de mettre en œuvre un processus de développement sécurisé. Il est à la fois open, indépendant du fournisseur et librement accessible à tous.

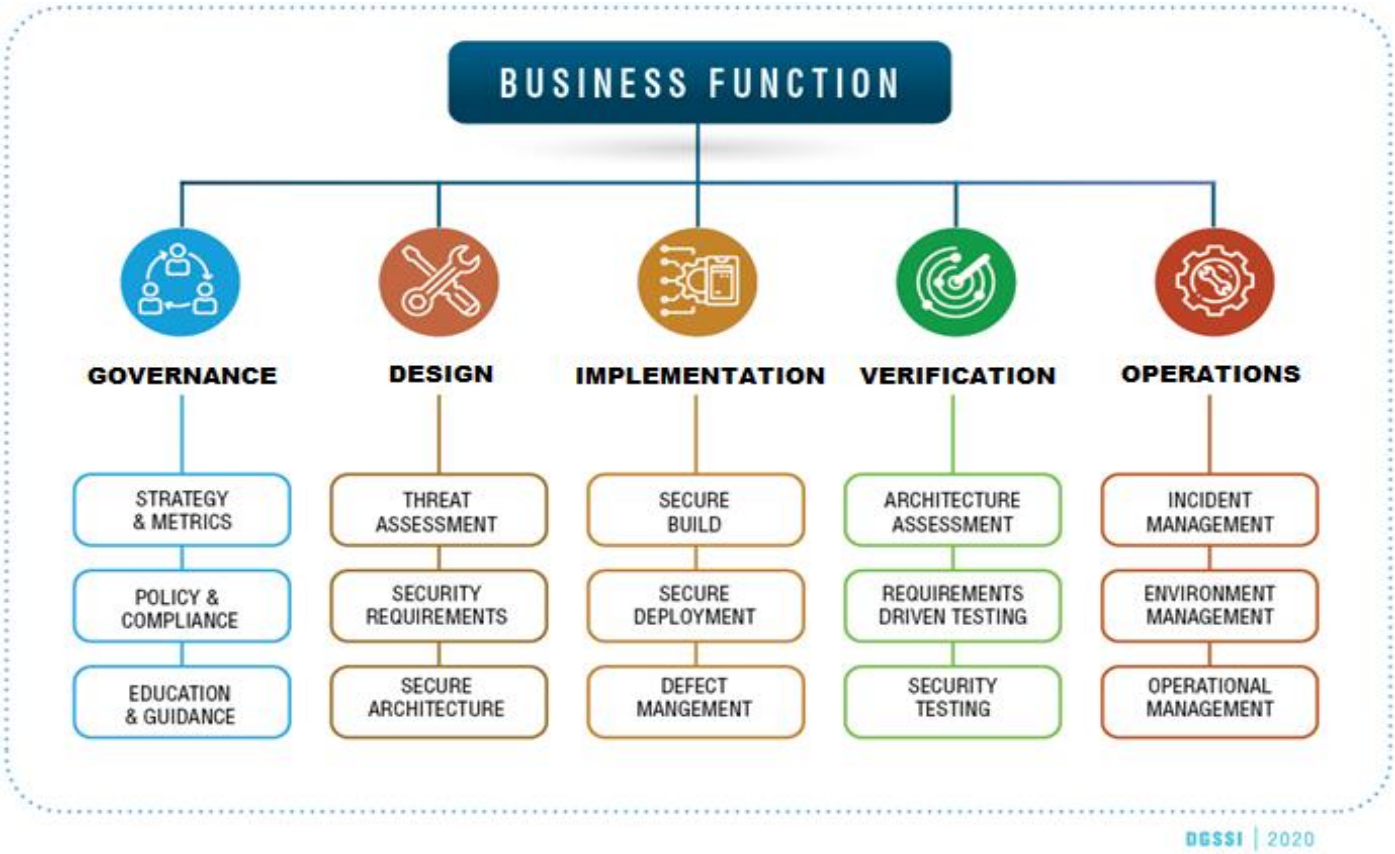


Figure 16 Fonctions métier SAMM 2.0

Il s'agit d'un cadre ouvert destiné à aider les organisations à formuler et à mettre en œuvre une stratégie de sécurité logicielle adaptée aux risques spécifiques auxquels chaque entreprise est confrontée. Les ressources fournies par SAMM aident à :

- **Évaluer les pratiques de la sécurité logicielle existantes d'une organisation ;**
- **Construire un programme équilibré de la sécurité logicielle dans des itérations bien définies ;**
- **Démontrer des améliorations concrètes à un programme d'assurance de la sécurité ;**
- **Définir et mesurer les activités liées à la sécurité au sein d'une organisation.**

SAMM a été défini avec une flexibilité à l'esprit permettant son utilisation par les petites, les moyennes et les grandes organisations quel que soit le style de développement qu'elle utilise. De plus, ce modèle peut être appliqué à l'échelle de toute l'organisation, pour un seul secteur d'activité ou même pour un projet individuel.

En étudiant le graphique (voir figure 17), représentant un cas exemple, nous pouvons constater que la plus grande progression de maturité concerne les deux pratiques de sécurité : la stratégie et les tests de sécurité. En effet, au niveau de la première phase le niveau été respectivement, environ, 0,75 et 0,5. Par ailleurs, au niveau de la phase 4, le niveau de maturité a atteint, pour les mêmes pratiques de sécurité, 2,1 et 2,6.

[16] SAMM: Software Assurance Maturity Model
[17] OWASP: Open Web Application Security Project

Il est important de signaler qu'une partie prenante qui souhaite utiliser ce modèle de maturité pour améliorer son programme d'assurance logicielle pourrait être intéressée par un seul principe de sécurité par exemple, une institution financière pourrait être intéressée par le niveau de maturité de la conformité et de la politique, car elle opère une industrie hautement réglementée.

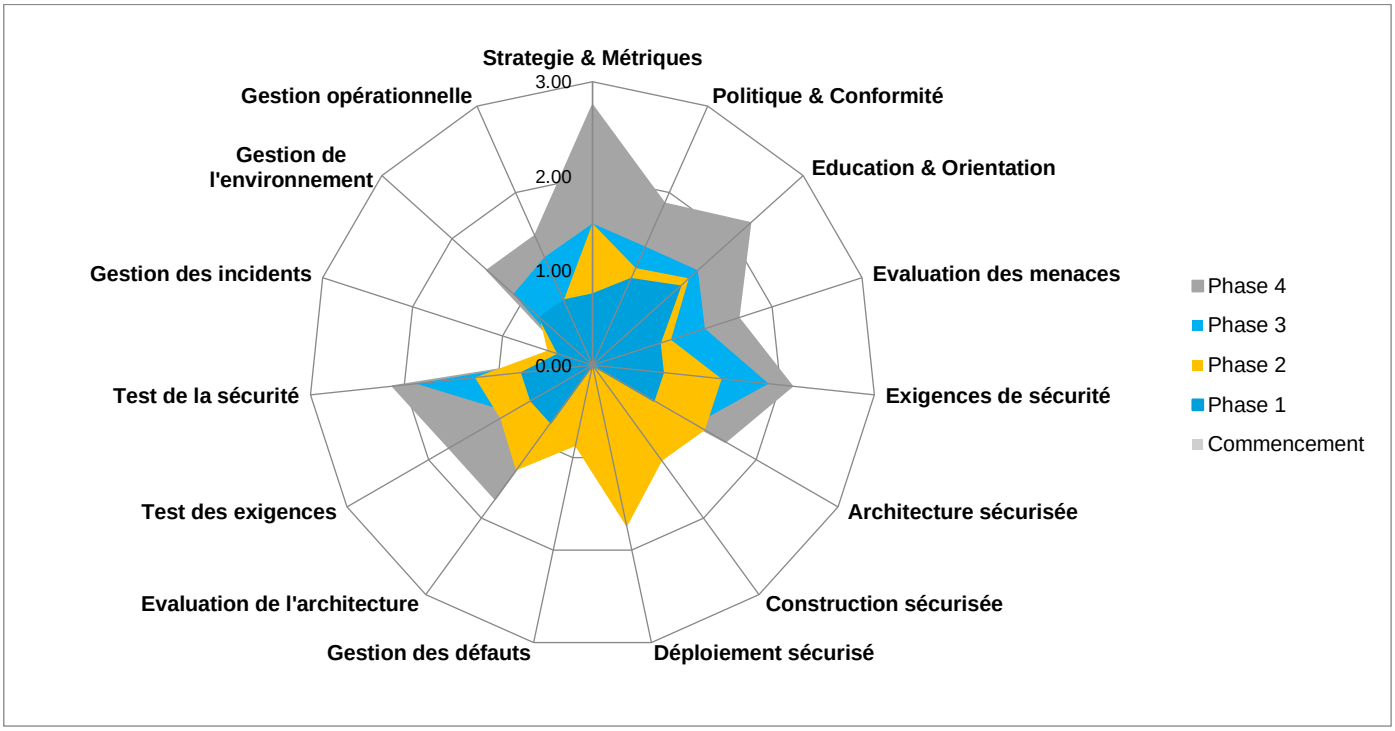


Figure 17 Niveaux de maturité SAMM 2.0

4.7 Synthèse

L'analyse et la comparaison des modèles actuels de développement de logiciels sécurisés est particulièrement importante. Les modèles classiques de développement logiciel adoptent une approche réactive, où les tâches de sécurité sont reléguées aux étapes finales du cycle de vie du logiciel. Néanmoins, les modèles de développement logiciel devraient intégrer des fonctionnalités et des réponses préventives aux problèmes de sécurité.

La figure 18 mets en évidence les similitudes et les différences entre les différents modèles étudiés :

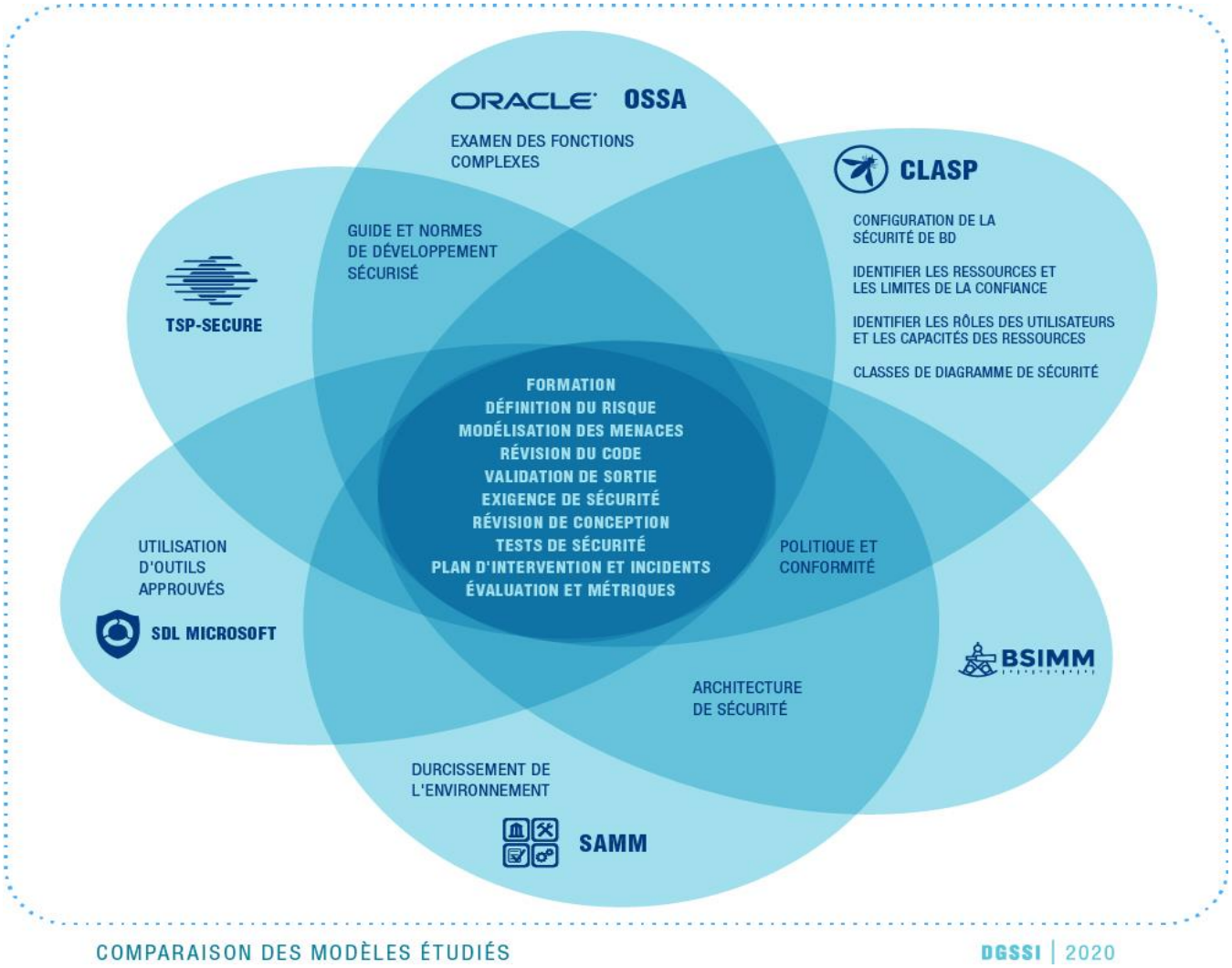


Figure 18 Comparaison des modèles étudiés

En général, les sociétés de développement génèrent des logiciels en fonction des exigences fonctionnelles. Lors de la phase de test du cycle de vie du développement, la qualité du logiciel est déterminée et les problèmes de sécurité sont identifiés et résolus. Le choix de la méthodologie de développement pourrait poser plusieurs problèmes et, en outre, l'estimation des efforts pour développer des logiciels sécurisés revêt une grande importance pour les éditeurs de logiciels. À cet égard, le NIST[18] affirme que résoudre une vulnérabilité pendant la phase de post-production est jusqu'à 30 fois plus cher que la résoudre pendant l'étape de collecte des exigences.

Les coûts augmentent de façon exponentielle lorsque les tâches de détection de vulnérabilité sont reléguées à la phase de test. Les mesures réactives peuvent aider à éviter les vulnérabilités, mais elles ne minimisent pas les coûts de sa résolution et ne protègent pas correctement les systèmes. En effet, l'identification des vulnérabilités - sans connaître leur origine - pourrait entraîner une refonte et des changements excessifs dans la mise en œuvre du logiciel.

La partie centrale du diagramme représente les similitudes des modèles de développement de logiciels sécurisés.

La comparaison montre que des modèles partagent *des* éléments en commun, notamment la définition des risques, la modélisation des menaces et les tests de sécurité. De même, d'autres modèles convergent vers les activités liées aux politiques et à la sécurisation de l'environnement de développement.

Le tableau 2 présente les activités de sécurité de ces modèles en fonction de leur cycle de vie. Comme on peut le voir, les activités de sécurité sont incluses (ou non) et mises en œuvre dans les modèles de différentes manières.

Ces méthodologies prennent en compte les activités de sécurité qui couvrent l'ensemble du processus de développement logiciel.

[18] NIST: National Institute of Standards and Technology

Phase	Microsoft SDL	OSSA	CLASP	TSP-Secure	SAMM	BSIMM
Stratégies			Identify global security policy Identify resources and boundaries of trust Identify user roles and resource capabilities Specify operational environment		Policy and Compliance	Policy and Compliance
Formation	Core Security Training	Training	Institute security awareness program	Training	Education and Guidance	Training
Exigences	Security and Privacy Risk Assessment Establish Security Requirements	Definition of risk Security Requirements	Identify attack surface Document security-relevant requirements	Risk analysis Security Requirements	Security Requirements Secure Architecture	Standards and Requirement Architecture Analysis
Conception	Threat Modelling Analyze Attack Surface Establish Design Requirements	Threat Modelling Design Revision Cases of misuses	Threat Modelling Apply security principles to design Security diagram classes Cases of misuse	Threat Modelling Security Design Cases of misuses	Threat assessment Design Review	Attack Models Security Features and Design
Implémentation	Use Approved Tools Deprecate Unsafe Functions	Code Revision Guide and safe development standards Review of complex functions	Security configuration in BD Integrate security analysis into source management process	Code Revision Guide and security development standards	Code Review	Code Review
Tests	Dynamic Analysis Fuzz testing	Security Testing	Security Testing	Security Testing Fuzz testing	Security Testing	Penetration Testing
Pre-Release	Final Security Review Release Archive	Security release checklists	Verify security attributes of resources			
Post-Release	Execute Incident Response Plan	Vulnerability Correction	Manage security issue disclosure process	Vulnerability Management	Vulnerability Management Environment Hardening Operational Enablement	Configuration Management and Vulnerability Management Software Environment
Métrique	Quality Gates and BUG BARS	Product security acquisition checklists	Monitor security metrics	Security metrics	Strategy and Metrics	Strategy and Metrics

Tableau 2 Activités de sécurité en fonction du SDLC

5 Modèle de maturité SAMM

SAMM est le cadre OWASP destiné à aider les organisations à évaluer, formuler et mettre en œuvre une stratégie de sécurité logicielle, qui peut être intégrée dans leur cycle de vie de développement logiciel SDLC existant.

La DGSSI recommande le SAMM 2.0 parce qu'il convient à la plupart des contextes - que l'organisation développe ses propres logiciels, sous-traite leur développement ou même si elle se concentre sur l'acquisition de ses logiciels, et quelle que soit la méthodologie SDLC qu'elle utilise.

Le SAMM 2.0 est conçu pour être indépendant du paradigme de développement, ce qui signifie qu'il peut être appliqué à tous types de projets allant de la cascade à l'Agile en passant par DevOps ainsi que certains modèles encore à concevoir.

5.1 Niveaux de maturité

Comme mentionné dans les paragraphes ci-après, SAMM 2.0 a quinze principes ou pratiques de sécurité. Chacun d'eux est divisé en deux flux, et chaque flux a un objectif.

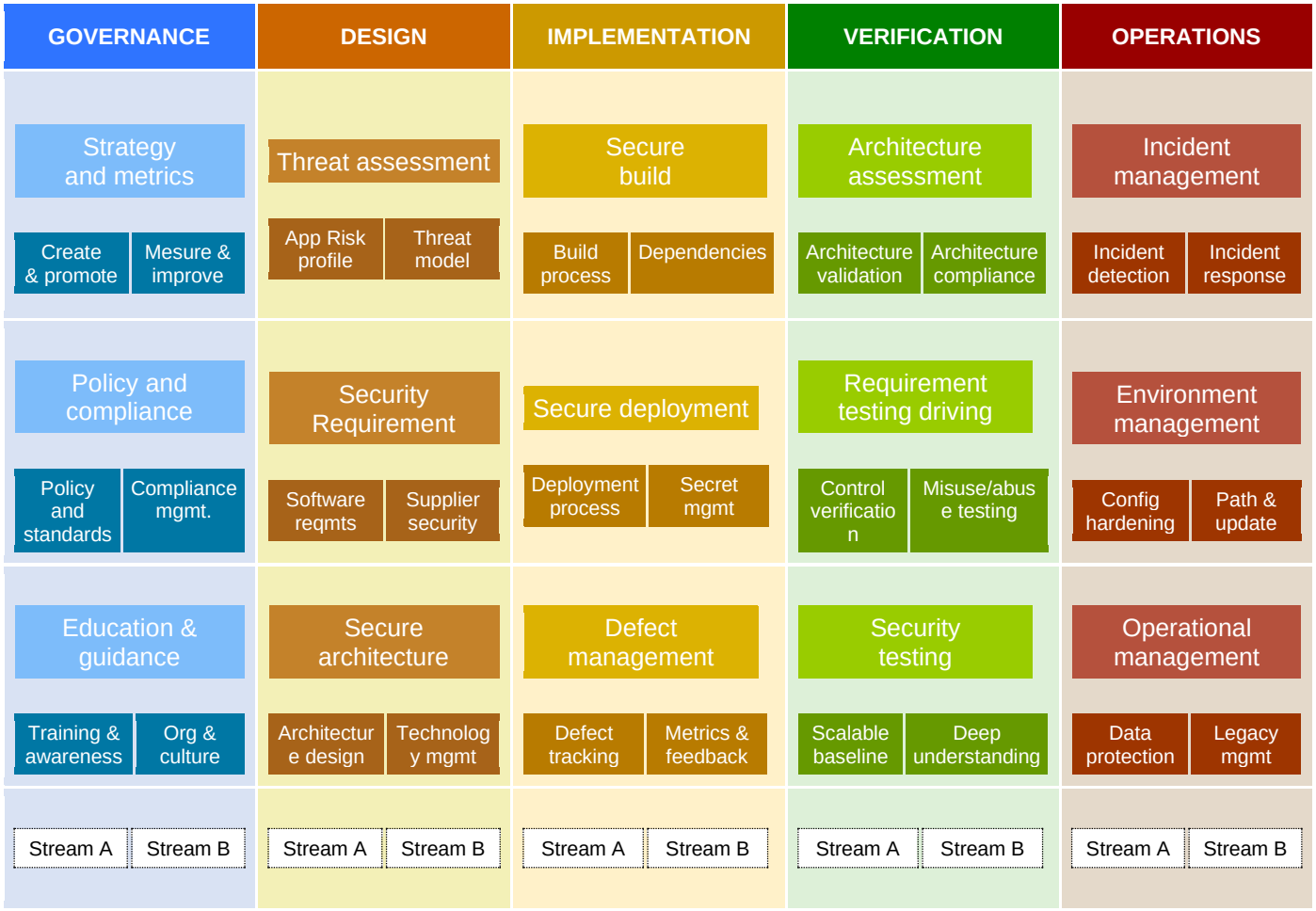


Figure 19 Flux géré par le modèle SAMM 2.0

Un objectif spécifique à un flux peut être atteint à différents niveaux de maturité :

- **Niveau 0 : un point de départ implicite avec la pratique non réalisée.**
- **Niveau 1 : une compréhension initiale et une mise en œuvre ad hoc de la pratique.**
- **Niveau 2 : réalisation structurée avec une efficience et une efficacité accrue de la pratique.**
- **Niveau 3 : une maîtrise complète de la pratique à une échelle qui s'accompagne d'un fonctionnement optimisé.**

Entre autres, pour chaque niveau, SAMM définit un objectif, un ensemble d'activités et décrit les résultats attendus.

MATURITY LEVELS		ASSESSMENT SCORES	
3	Comprehensive mastery at scale	**1**	Most
2	Increased efficiency and effectiveness	**0.5**	At least half
1	Ad-hoc provision	**0.2**	Some
0	Practice unfulfilled	**0**	None

Tableau 3 Niveaux de maturité et notation pour le modèle SAMM 2.0

Au plus haut niveau, SAMM définit cinq fonctions métier critiques. Chaque fonction métier est une catégorie d'activités liées aux rouages du développement logiciel, ou autrement dit, toute organisation impliquée dans le développement logiciel doit remplir chacune de ces fonctions métier dans une certaine mesure.

Pour chaque fonction métier, SAMM définit trois pratiques de sécurité. Chaque pratique de sécurité est un domaine d'activités liées à la sécurité qui renforcent l'assurance de la fonction métier associée. Il existe quinze pratiques de sécurité qui sont des silos indépendants qui correspondent aux cinq fonctions métier du développement logiciel.

Pour chaque pratique de sécurité, SAMM définit trois niveaux de maturité comme objectifs. Chaque niveau au sein d'une pratique de sécurité est caractérisé par un objectif de plus en plus sophistiqué défini par des activités spécifiques et des mesures de réussite plus strictes que le niveau précédent. De plus, chaque pratique de sécurité peut être améliorée indépendamment, bien que les activités associées puissent conduire à des optimisations.

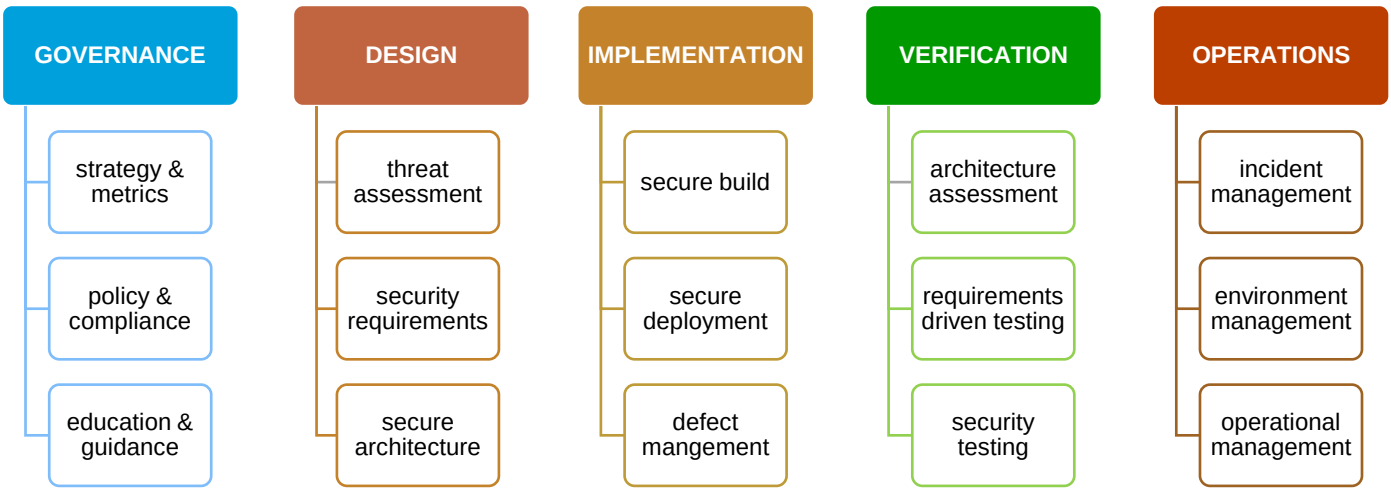


Figure 20 Fonctions métier SAMM 2.0

5.2 Governance

La gouvernance est focalisée sur les processus et les activités liés à la façon dont une organisation gère les activités globales de développement de logiciels. Plus spécifiquement, cela inclut les préoccupations qui ont un impact sur les groupes inter fonctionnels impliqués dans le développement, ainsi que sur les processus métier établis au niveau de l'organisation.

5.2.1 Strategy & Metrics

La pratique Strategy & Metrics SM[19] se concentre sur l'établissement d'un cadre au sein d'une organisation pour un programme d'assurance de la sécurité logicielle. Il s'agit de l'étape la plus fondamentale de la définition des objectifs de sécurité d'une manière à la fois mesurable et alignée sur le risque métier réel de l'organisation.

En commençant par des profils de risque légers, une organisation évolue vers des schémas de classification des risques plus avancés pour les actifs d'application et de données au fil du temps. Avec des informations supplémentaires sur les mesures de risque, une organisation peut ajuster ses objectifs de sécurité au niveau de ses projets et développer des feuilles de route granulaires pour rendre le programme de sécurité plus efficace. Aux niveaux les plus avancés de cette pratique, une organisation s'appuie sur de nombreuses sources de données, internes et externes, pour collecter des

[19] SM: Strategy and Metrics

mesures et des commentaires qualitatifs sur le programme de sécurité. Cela permet d'ajuster avec précision les dépenses par rapport à l'avantage réalisé au niveau du programme.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Créer et promouvoir**	**Mesurer et améliorer**
1	Identifier les objectifs et les moyens pour mesurer l'efficacité du programme de sécurité.	Identifier les moteurs de l'organisation en fonction de la tolérance au risque de l'organisation.	Définissez des mesures avec un aperçu de l'efficacité et de l'efficience du programme de sécurité des applications.
2	Établir une feuille de route stratégique unifiée pour la sécurité des logiciels au sein de l'organisation.	Publiez une stratégie unifiée pour la sécurité des applications.	Fixez des objectifs et des indicateurs de performance clés pour mesurer l'efficacité du programme.
3	Aligner les efforts de la sécurité sur les indicateurs organisationnels et les valeurs des actifs pertinents.	Alignez le programme de sécurité des applications pour soutenir la croissance de l'organisation.	Influencer la stratégie en fonction des métriques et des besoins organisationnels.

Tableau 4 Strategy & Metrics - Niveaux de maturité

5.2.2 Policy & Compliance

La pratique Policy and Compliance PC[20] est axée sur la compréhension et le respect des exigences juridiques et réglementaires externes, tout en établissant des normes de sécurité internes pour garantir la conformité d'une manière qui est alignée avec l'objectif métier de l'organisation.

L'un des thèmes d'amélioration de cette pratique est de se concentrer sur les audits au niveau du projet qui recueillent des informations sur le comportement de l'organisation afin de vérifier que les attentes sont satisfaites. En introduisant des audits de routine qui commencent légers et se développent en profondeur au fil du temps, le changement organisationnel est réalisé de manière itérative.

Sous une forme sophistiquée, la fourniture de cette pratique implique une compréhension à l'échelle de l'organisation des normes internes et des facteurs de conformité externes tout en maintenant des points de contrôle à faible latence avec les équipes de projet pour s'assurer qu'aucun projet ne fonctionne en dehors des attentes fixées.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveaux	**Description**	**Politique et normes**	**Compliance Management**
1	Identifier et documenter les moteurs de gouvernance et de conformité pertinents pour l'organisation.	Déterminer une base de sécurité représentant les politiques et les normes de l'organisation.	Identifiez les moteurs et exigences de conformité tiers et établissez une correspondance avec les politiques et normes existantes.
2	Établissez une base de sécurité et de conformité spécifique à l'application.	Développer les exigences de sécurité applicables à toutes les applications.	Publier des exigences d'application spécifiques à la conformité et des conseils de test.
3	Mesurer le respect des politiques, des normes et des exigences des tiers.	Mesurer et rendre compte de l'état de l'adhésion de chaque application aux politiques et aux normes.	Mesurer et rendre compte de la conformité de chaque application aux exigences des tiers.

Tableau 5 Policy & Compliance - Niveaux de maturité

5.2.3 Education & Guidance

La pratique Education & Guidance EG[21] est axée sur l'armement du personnel impliqué dans le cycle de vie du logiciel avec des connaissances et des ressources pour concevoir, développer et déployer des logiciels sécurisés. Grâce à un meilleur accès aux informations, les équipes de projet seront en mesure d'identifier et d'atténuer de manière proactive les risques de sécurité spécifiques qui s'appliquent à leur organisation.

L'un des principaux thèmes d'amélioration des objectifs est la formation des employés, que ce soit par le biais de sessions animées par un instructeur ou de modules sur ordinateur. Au fur et à mesure que l'organisation progresse, une large base de formation est construite en commençant par les développeurs et en passant à d'autres rôles dans toute l'organisation, culminant avec l'ajout d'une certification basée sur les rôles pour assurer la compréhension du matériel.

[20] PC: Policy and Compliance
[21] EG: Education and Guidance

Outre la formation, cette pratique nécessite également d'intégrer des informations relatives à la sécurité dans des lignes directrices qui servent d'informations de référence au personnel. Cela crée une base pour les pratiques de sécurité dans l'organisation, et permet ultérieurement une amélioration incrémentielle une fois que l'utilisation des directives a été adoptée.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Formation et sensibilisation**	**Organisation & Culture**
1	Offrir au personnel un accès à des ressources autour des thèmes du développement et du déploiement sécurisés.	Fournir une formation de sensibilisation à la sécurité pour tout le personnel impliqué dans le développement de logiciels.	Identifier un "champion de la sécurité" au sein de chaque équipe de développement.
2	Éduquez tout le personnel dans le cycle de vie du logiciel avec la technologie et des conseils spécifiques au rôle sur le développement sécurisé.	Offrir des conseils spécifiques à la technologie et au rôle, y compris les nuances de sécurité de chaque langue et plateforme.	Développer un centre d'excellence logiciel sécurisé favorisant le leadership éclairé parmi les développeurs et les architectes.
3	Développer des programmes de formation internes animés par des développeurs de différentes équipes.	Conseils internes standardisés concernant les normes de développement de logiciels sécurisés de l'organisation.	Construire une communauté logicielle sécurisée comprenant toutes les personnes de l'organisation impliquées dans la sécurité des logiciels.

Tableau 6 Education & Guidance – Niveaux de maturité

5.3 Design

5.3.1 Threat Assessment

La pratique Threat Assessment TA[22] se base sur l'identification et la compréhension des risques au niveau du projet en fonction de la fonctionnalité du logiciel en cours de développement et des caractéristiques de l'environnement d'exécution.

En se basant sur les détails relatifs aux menaces et aux attaques probables contre chaque projet, l'organisation dans son ensemble fonctionne plus efficacement grâce à de meilleures décisions concernant la hiérarchisation des initiatives de la sécurité. De plus, les décisions d'acceptation des risques sont plus éclairées, donc mieux alignées sur l'activité.

En commençant par des modèles de menaces simples et en développant des méthodes plus détaillées d'analyse et de pondération des menaces, une organisation s'améliore au fil du temps. Il est à préciser qu'une organisation sophistiquée conserverait ces informations d'une manière étroitement liée aux facteurs de compensation et aux risques de transmission des entités externes.

Cela permet de mieux comprendre les impacts potentiels en aval des problèmes de sécurité tout en surveillant de près les performances actuelles de l'organisation par rapport aux menaces connues.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Profil de risque d'application**	**Modélisation des menaces**
1	Identification au mieux des menaces de haut niveau pour l'organisation et les projets individuels.	Évaluation de base du risque d'application.	Modélisation ad hoc des menaces au mieux.
2	Standardisation et analyse à l'échelle de l'entreprise des menaces liées aux logiciels au sein de l'organisation.	Comprendre le risque pour toutes les applications de l'organisation.	Modélisation standardisée des menaces.
3	Amélioration proactive de la couverture des menaces dans toute l'organisation.	Examiner périodiquement les profils de risque des applications.	Améliorer la qualité grâce à une analyse automatisée.

Tableau 7 Threat assessment – Niveaux de maturité

[22] TA: Threat Assessment

5.3.2 Security Requirements

La pratique Security Requirements SR[23] est axée sur la spécification proactive du comportement attendu des logiciels en matière de sécurité. Grâce à l'ajout d'activités d'analyse au niveau du projet, les exigences de sécurité sont initialement rassemblées en fonction de l'objectif métier de haut niveau du logiciel.

Au fur et à mesure que l'organisation progresse, des techniques plus avancées sont utilisées, telles que les spécifications de contrôle d'accès, pour découvrir de nouvelles exigences de sécurité qui n'étaient peut-être pas évidentes au départ pour le développement. Sous une forme sophistiquée, la fourniture de cette pratique implique également de pousser les exigences de sécurité de l'organisation dans ses relations avec les fournisseurs, puis d'auditer les projets pour s'assurer que tous respectent les attentes en matière de spécification des exigences de sécurité.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Configuration logicielle requise**	**Sécurité des fournisseurs**
1	Tenir compte de la sécurité de manière explicite pendant le processus des exigences logicielles	Objectifs de sécurité des applications de haut niveau.	Évaluer le fournisseur en fonction de la sécurité
2	Augmenter la granularité des exigences de sécurité dérivées de la logique métier et des risques connus	Ingénierie des exigences structurées	Intégrez la sécurité dans les accords avec les fournisseurs
3	Processus d'exigences de sécurité du mandat pour tous les projets logiciels et dépendances tierces	Construire un cadre d'exigences standard	Assurer une couverture adéquate pour les fournisseurs externes

Tableau 8 Security requirements - Niveaux de maturité

5.3.3 Secure Architecture

La pratique de l'architecture sécurisée SA[24] se concentre sur les étapes proactives permettant à une organisation de concevoir et de créer un logiciel sécurisé par défaut. En améliorant le processus de conception de logiciels avec des services et des composants réutilisables, le risque de sécurité global lié au développement de logiciels peut être considérablement réduit.

En partant de simples recommandations sur les Framework logiciels et d'une prise en compte explicite des principes de conception sécurisée, une organisation évolue vers une utilisation cohérente des modèles de conception pour les fonctionnalités de sécurité. En outre, les activités encouragent les équipes de projet à utiliser davantage les services et les infrastructures de sécurité centralisés.

À mesure qu'une organisation évolue au fil du temps, l'adoption de cette pratique implique la création des plates-formes de référence pour couvrir les types génériques de logiciels qu'elle crée. Celles-ci servent de Framework sur lesquels les développeurs peuvent créer des logiciels personnalisés avec un risque moindre de vulnérabilités.

[23] SR: Security Requirements
[24] SA: Secure Architecture

Niveaux de maturité

Niveaux de maturité		Flux & objectif par niveau	
Niveau	**Description**	**Architecture Design**	**Gestion de la technologie**
1	Intégrer la prise en compte des conseils de sécurité proactifs dans le processus de conception du logiciel	Utiliser les principes de sécurité de base	Obtenir des technologies, des cadres et des intégrations dans la solution globale
2	Orienter le processus de conception logicielle vers des services sécurisés connus et des conceptions sécurisées par défaut	Établir des modèles de conception et des solutions de sécurité communes	Standardiser les technologies et les cadres à utiliser dans les différentes applications
3	Contrôler formellement le processus de conception du logiciel et valider l'utilisation des composants sécurisés.	Créer des architectures de référence	Imposer l'utilisation de technologies standards sur tout développement logiciel

Tableau 9 Secure architecture - Niveaux de maturité

5.4 Implementation

5.4.1 Secure Build

Il est important de pouvoir produire des builds reproductibles de manière cohérente, et ceci est mieux réalisé en ayant un environnement de build automatisé basé sur des composants logiciels et des services fiables. Moins une construction nécessite de travail manuel, mieux c'est. Un pipeline de construction automatisé peut effectuer des tests de non régression et d'intégration automatisés ainsi que des mécanismes de test de sécurité d'application statique et dynamique (SAST[25], DAST[26]).

Niveaux de maturité

Niveaux de maturité		Flux & objectif par niveau	
Niveau	**Description**	**Processus de construction**	**Dépendances logicielles**
1	Le processus de construction est répétable et cohérent	Le processus de construction est défini et cohérent	Toutes les dépendances d'application sont identifiées et documentées
2	Le processus de construction est optimisé et entièrement intégré au flux de travail	Le processus de construction est entièrement automatisé et ne nécessite aucune intervention du développeur	Tous les composants et bibliothèques sont périodiquement examinés pour les vulnérabilités de sécurité connues et les problèmes de licence
3	Le processus de construction aide à empêcher les défauts connus d'entrer dans l'environnement de production	Les défauts de sécurité peuvent déclencher l'arrêt de l'exécution de la compilation	Les composants et les bibliothèques sont analysés indépendamment pour les vulnérabilités

Tableau 10 Secure build - Niveaux de maturité

[25] SAST: Static Application Security Testing
[26] DAST: Dynamic Application Security Testing

5.4.2 Secure Deployment

S'il est essentiel de prendre en compte les aspects de sécurité lors de la conception et de la mise en œuvre du logiciel, le même travail doit également être effectué lors du déploiement. Cela commence par automatiser le processus du déploiement autant que possible afin de réduire (voir supprimer complètement) les risques d'erreurs humaines et de maintenir des processus constamment répétés avec les vérifications et les journaux nécessaires. De même, sinon plus, le processus de déploiement doit protéger les fondements de la confidentialité et maintenir le contrôle des données sensibles telles que les clés privées, les mots de passe et autres jetons, ainsi que l'intégrité des stockages de données.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Processus de déploiement**	**Secret Management**
1	Les processus de déploiement sont entièrement documentés	Le déploiement est automatisé ou effectué par une personne autre que le développeur	Les secrets de production sont cryptés et non gérés par les développeurs
2	Les processus de déploiement incluent des jalons de vérification de sécurité	Intégration de la vérification de sécurité dans le déploiement (par exemple, analyse de code statique binaire / analyse AV)	Les secrets sont inclus de manière dynamique pendant le processus de déploiement
3	Le processus de déploiement est entièrement automatisé et intègre une vérification automatisée de tous les jalons critiques	L'intégrité du code est vérifiée avant le déploiement	Les fichiers et les référentiels sont vérifiés périodiquement pour les secrets qui doivent être protégés

Tableau 11 Secure Deployment - Niveaux de maturité

5.4.3 Defect Management

Il n'existe pas de logiciel parfait qui n'a pas de bugs ; ne pas avoir de problèmes devrait être une source de préoccupation plutôt que de soulagement car il est presque garanti que quels que soient les problèmes que les tests pourraient manquer, ceux-ci seront découverts en production. Les défauts et les logiciels vont de pair, il est donc avantageux d'établir un processus contrôlé pour collecter, enregistrer et analyser les défauts (côté sécurité) des logiciels pour permettre une prise de décision basée sur des mesures.

Niveaux de maturité

Niveaux de maturité		Flux & objectif par niveau	
Niveau	**Description**	**Suivi des défauts**	**Métriques et commentaires / Apprentissage**
1	Tous les défauts sont suivis dans chaque projet	Suivre tous les défauts	Calculer et partager des statistiques de base, telles que le nombre total
2	Suivi des défauts utilisé pour influencer le processus de déploiement	Attribuer une note de sécurité basée sur le SLA au défaut	Calculez des mesures plus avancées qui incluent la vitesse des nouveaux problèmes, les mesures de vitesse de correction et les tendances.
3	Suivi des défauts sur plusieurs composants est utilisé pour réduire le nombre de nouveaux défauts	Mesurer et appliquer le respect du SLA	Utilisez l'analyse des tendances pour influencer les changements dans la phase de conception et de mise en œuvre sur plusieurs projets.

Tableau 12 Defect management - Niveaux de maturité

5.5 Verification

5.5.1 Architecture Assessment

La pratique Architecture Assessment est axée sur l'évaluation de la conception et de l'architecture des logiciels pour les problèmes liés à la sécurité. Cela permet à une organisation de détecter les problèmes au niveau de l'architecture dès le début du développement logiciel et d'éviter ainsi des coûts potentiellement importants liés à la refactorisation ultérieure en raison de problèmes liés à la sécurité.

En commençant par des activités légères pour développer la compréhension des détails relatifs à la sécurité d'une architecture, une organisation évolue vers des méthodes d'inspection plus formelles qui vérifient l'exhaustivité de la fourniture de mécanismes de sécurité. Au niveau de l'organisation, des services de revue de conception sont élaborés et proposés aux parties prenantes.

Sous une forme sophistiquée, la fourniture de cette pratique implique une inspection détaillée, au niveau des données, des conceptions et l'application des attentes de base pour la conduite des évaluations de conception et l'examen des résultats avant que les versions ne soient acceptées.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Architecture Validation**	**Architecture Compliance**
1	Revoir l'architecture pour s'assurer que des mesures d'atténuation de base sont en place pour les risques connus	Identifier les composants d'architecture d'application et d'infrastructure	Examen ad hoc de l'architecture par rapport aux exigences de conformité
2	Revoir la fourniture complète des mécanismes de sécurité dans l'architecture	Valider les mécanismes de sécurité de l'architecture	Analyser l'architecture par rapport aux exigences de sécurité connues et aux meilleures pratiques
3	Examiner l'efficacité de l'architecture et les résultats des commentaires pour améliorer l'architecture de sécurité	Examiner l'efficacité des composants de l'architecture	Rétroaction des résultats de l'examen de l'architecture dans l'architecture d'entreprise, les principes et modèles de conception de l'organisation, les solutions de sécurité et l'architecture de référence

Tableau 13 Architecture assessment - Niveaux de maturité

5.5.2 Requirements driven testing

La pratique Requirements driven testing se concentre sur l'inspection des logiciels au niveau du code source et de son versioning afin de détecter les failles de sécurité. Les vulnérabilités au niveau du code sont généralement simples à comprendre sur le plan conceptuel, mais même les développeurs avertis peuvent facilement commettre des erreurs qui exposent le logiciel à des compromis potentiels.

Pour commencer, une organisation utilise des listes de contrôle légères et, pour plus d'efficacité, n'inspecte que les modules logiciels les plus critiques. Cependant, à mesure qu'une organisation évolue, elle utilise la technologie d'automatisation pour améliorer considérablement la couverture et l'efficacité des activités d'examen de la mise en œuvre.

Dans sa forme sophistiquée, cette pratique implique une intégration plus approfondie de l'examen de la mise en œuvre dans le processus de développement pour permettre aux équipes de projet de détecter les problèmes plus tôt. Cela permet également aux organisations de mieux auditer et de définir les attentes concernant les résultats de l'examen de la mise en œuvre avant que les publications ne puissent être effectuées.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Vérification de contrôle**	**Tests des mal-utilisations et des**
1	Trouver de manière opportuniste les vulnérabilités de base et autres problèmes de sécurité	Tester les contrôles de la sécurité standard	Effectuer des tests de fuzzing de sécurité
2	Effectuer un examen de l'implémentation pour découvrir les risques spécifiques à l'application par rapport aux exigences de sécurité	Dériver des cas de test à partir des exigences de sécurité connues	Créer et tester des cas d'abus et des tests de failles de logique métier
3	Maintenir le niveau de sécurité de l'application après les corrections de bogues, les modifications et / ou pendant la maintenance	Effectuer des tests de régression (avec des tests unitaires de sécurité)	Déni de service et tests de résistance à la sécurité

Tableau 14 Requirements driven testing - Niveaux de maturité

5.5.3 Security Testing

La pratique Security Testing ST[27] se concentre sur l'inspection des logiciels dans l'environnement d'exécution afin de détecter les problèmes de sécurité. Ces activités de test renforcent le cas d'assurance du logiciel en le vérifiant dans le même contexte dans lequel il est censé s'exécuter, rendant ainsi visibles les erreurs de configuration opérationnelle ou les erreurs de logique métiers difficiles à trouver autrement.

En commençant par les tests de pénétration et les cas de test de haut niveau basés sur la fonctionnalité du logiciel, une organisation évolue vers l'utilisation de l'automatisation des tests de sécurité pour couvrir la grande variété de cas de test qui pourraient démontrer une vulnérabilité dans le système.

Dans une forme avancée, la fourniture de cette pratique implique la personnalisation de l'automatisation des tests pour créer une batterie de tests de sécurité couvrant en détail les préoccupations spécifiques à l'application. Avec une visibilité supplémentaire au niveau de l'organisation, les tests de sécurité permettent aux organisations de définir des attentes minimales pour les résultats des tests de sécurité avant qu'une version de projet soit acceptée.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Ligne de base évolutive**	**Compréhension profonde**
1	Effectuer des tests de sécurité (manuels et basés sur des outils) pour découvrir les défauts de sécurité	Utiliser des outils de test de sécurité automatisés	Effectuer des tests de sécurité manuels des composants à haut risque
2	Rendre les tests de sécurité pendant le développement plus complets et plus efficaces grâce à une automatisation complétée par des tests de pénétration de sécurité manuels réguliers	Utiliser l'automatisation des tests de sécurité spécifiques aux applications	Effectuer des tests de pénétration manuels
3	Intégrer les tests de sécurité dans le cadre des processus de développement et de déploiement	Intégrez des tests de sécurité automatisés dans le processus de création et de déploiement	Intégrer les tests de sécurité dans le processus de développement

Tableau 15 Security testing - Niveaux de maturité

[27] ST: Security Testing

5.6 Operations

5.6.1 Incident Management

La pratique Incident Management IM[28] est axée sur les processus au sein d'une organisation en ce qui concerne le traitement des rapports des problèmes et des incidents opérationnels. En ayant ces processus en place, les projets d'une organisation auront des attentes cohérentes et une efficacité accrue pour gérer ces événements, plutôt que des réponses chaotiques. À partir d'une attribution légère des rôles en cas d'incident, une organisation évolue vers un processus de réponse aux incidents plus formel qui assure la visibilité et le suivi des problèmes qui surviennent. Les communications sont également revues pour améliorer la compréhension globale des processus.

Sous une forme avancée, la gestion des problèmes implique une dissection approfondie des incidents et des rapports sur les problèmes afin de collecter des mesures détaillées et d'autres informations sur leurs causes profondes afin de faire un retour sur le comportement en aval de l'organisation.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Détection d'incidents**	**Réponse aux incidents**
1	Détection et traitement des incidents au meilleur effort	Détection d'incident au mieux avec les données de journal disponibles	Définition d'une stratégie de réponse aux incidents de haut niveau
2	Processus formel de gestion des incidents en place	Évaluation automatisée des journaux pilotée par le processus	Analyse des causes profondes avec une boucle de rétroaction
3	Gestion avancée des incidents	Gestion fiable des incidents en temps opportun	Incident proactif + exercices d'urgence

Tableau 16 Incident management - Niveaux de maturité

5.6.2 Environment Management

La pratique Environment Management EM[29] se concentre sur la création d'une assurance pour l'environnement d'exécution qui héberge les logiciels de l'organisation. Étant donné que le fonctionnement sécurisé d'une application peut être détérioré par des problèmes dans les composants externes, le renforcement de cette infrastructure sous-jacente améliore directement la posture de sécurité globale du logiciel.

En commençant par un simple suivi et en distribuant des informations sur l'environnement d'exploitation pour mieux informer les équipes de développement, une organisation évolue vers des méthodes évolutives pour gérer le déploiement des correctifs de sécurité et instrumenter l'environnement d'exploitation avec des détecteurs d'alerte précoce pour les problèmes de sécurité potentiels avant que les dommages ne soient causés.

Au fur et à mesure que l'organisation progresse, l'environnement d'exploitation est revu et renforcé par le déploiement d'outils de protection pour ajouter des couches de défenses et des filets de sécurité pour limiter les dommages en cas d'exploitation de vulnérabilités.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Durcissement de la configuration**	**Patch et mise à jour**
1	Correction et durcissement au meilleur effort	Durcissement au meilleur effort prioritaire	Correctif au meilleur effort prioritaire
2	Processus formel avec des références en place	Base de référence et directives de durcissement disponibles	Processus formel couvrant la pile complète
3	Conformité avec un processus d'amélioration continue appliquée	Détection et traitement des non-conformités	Consolidation du processus de mise à jour avec SLA et reporting

Tableau 17 Environment management - Niveaux de maturité

[28] IM: Incident Management
[29] EM: Environment Management

5.6.3 Operational Management

La pratique Operational Management se concentre sur la collecte d'informations critiques sur la sécurité auprès des équipes de projet qui créent des logiciels et de les communiquer aux utilisateurs et aux opérateurs du logiciel. Sans ces informations, même le logiciel le plus sécurisé comporte des risques excessifs, car les caractéristiques et choix de sécurité importants ne seront pas connus sur un site de déploiement.

En partant d'une documentation légère pour capturer les détails les plus importants pour les utilisateurs et les opérateurs, une organisation évolue vers la création de guides de sécurité opérationnelle complets qui sont fournis avec chaque version.

Sous une forme avancée, l'activation opérationnelle implique également des vérifications au niveau de l'organisation par rapport aux équipes de projet pour s'assurer que les informations sont capturées et partagées conformément aux attentes.

Niveaux de maturité

Niveaux de maturité		Flux et objectif par niveau	
Niveau	**Description**	**Protection des données**	**Décommissionnement d'un système / Gestion des héritages**
1	Pratiques fondamentales	Protections de base des données en place	Identification des applications / services inutilisés et hérités
2	Processus gérés et réactifs	Les données sont cataloguées et des politiques de protection des données sont établies	Des processus de déclassement et de migration hérités sont en place
3	Surveillance active et réponse	Les violations de la politique des données sont détectées et traitées	Gestion proactive et fiable des applications / services hérités

Tableau 18 Operational Management - Niveaux de maturité

5.7 Synthèse

La DGSSI propose à toutes les organisations de calculer leur niveau de maturité avec SAMM 2.0 : Faire un premier pas dans cette démarche est simple, il ne nécessite pas que toutes les actions soient mises en place. C'est un processus itératif, donc il est tout à fait normal de passer le premier et même le deuxième cycle à simplement apprendre les ficelles du métier.

6 Matrice de calcul du niveau de la maturité

L'approche typique de l'utilisation de la matrice de calcul de la maturité SAMM 2.0 dans une organisation est de commencer par la préparation, de passer par l'évaluation, de définir l'objectif, la planification et la mise en œuvre pour le déploiement. La matrice de calcul de la maturité SAMM 2.0 est particulièrement bien adapté pour soutenir l'amélioration continue, auquel cas le cycle est exécuté en continu, généralement par périodes de 3 à 12 mois. Notez qu'il n'est pas nécessaire de toujours exécuter toutes ces étapes. Vous pouvez utiliser la matrice du SAMM pour effectuer uniquement l'évaluation ou pour définir les objectifs à long terme, par exemple.

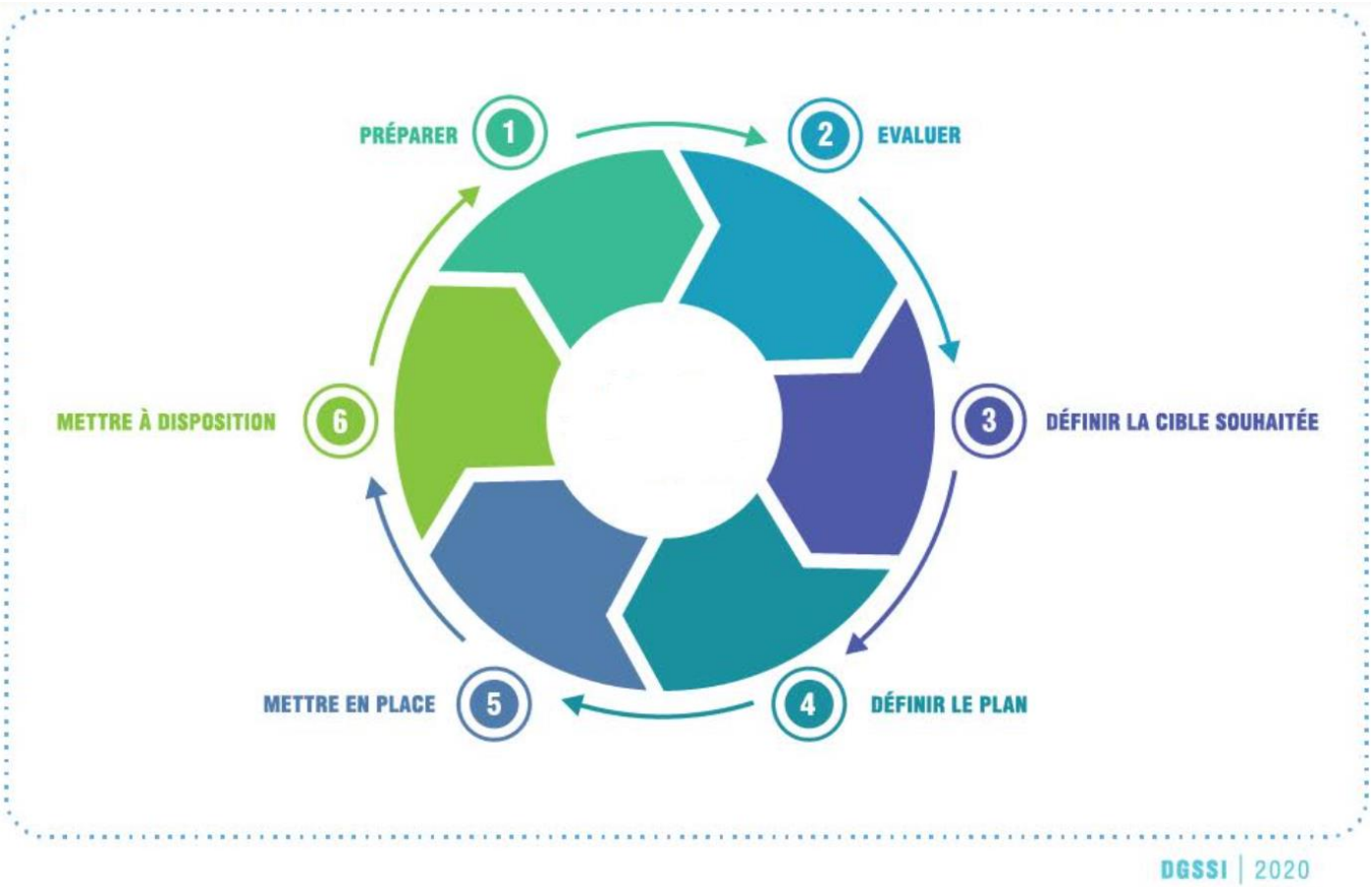


Figure 21 Phases de la mise en place du SAMM

Dans le cadre d'un effort de démarrage rapide, une seule personne peut exécuter les quatre premières phases (préparation, évaluation, fixation de l'objectif et définition du plan) dans un laps de temps limité (un à deux jours). S'assurer que cela est pris en charge dans l'organisation, ainsi que les phases de mise en œuvre et de déploiement, prend généralement beaucoup plus de temps.

6.1 Préparation

6.1.1 Déroulement

L'objectif principal de cette étape est d'assurer un bon démarrage du projet d'évaluation du niveau de la maturité de la sécurité au niveau d'une organisation. Afin de réussir cette étape de préparation, il faudrait :

- **Définir la cible de l'effort : toute l'organisation, une application ou un projet particulier, une équipe en particulier.**
- **S'assurer que les parties prenantes importantes sont identifiées et bien alignées pour soutenir le projet.**
- **Envisager d'impliquer au moins : le sponsor exécutif, l'équipe de sécurité, les développeurs, les architectes, les propriétaires d'entreprise, les testeurs d'assurance qualité et les gestionnaires.**

6.1.2 Recommandations

- **Pré-cribler la maturité du développement logiciel pour avoir des attentes réalistes.**
- **Commencer sur un périmètre réduit : Plus la portée est petite, plus l'exercice est facile.**

6.2 Evaluation

6.2.1 Déroulement

L'étape d'évaluation aide à identifier et à comprendre la maturité du périmètre choisi dans chacune des 15 pratiques de la sécurité logicielle décrites dans le chapitre « Modèle de maturité SAMM ».

Cette étape vise l'évaluation des pratiques actuelles via la disposition des entretiens avec les parties prenantes concernées pour comprendre l'état actuel des pratiques au sein d'une organisation. L'évaluation peut se faire par l'organisation elle-même à base de la matrice publiée sur le site web de la DGSSI.

A base du résultat de l'évaluation des pratiques actuelles, il faudrait déterminer pour chaque pratique de sécurité le niveau de maturité selon le système de notation de la maturité SAMM. Les activités sont notées par un système à choix multiples et font l'objet d'une moyenne pour le domaine de pratique de la sécurité, puis additionnées pour déterminer le score global.

6.2.2 Recommandations

- **Assurer une évaluation cohérente des différentes parties prenantes et équipes en utilisant les mêmes questions et intervieweur.**
- **Envisager d'utiliser différents formats pour recueillir des données, par exemple des ateliers ou des entretiens.**
- **S'assurer que les personnes interrogées comprennent les particularités des activités.**
- **Comprendre quelles activités ne sont pas applicables à l'organisation et en tenir compte dans la notation globale.**
- **Anticiper / documenter les divers appels à l'évaluation.**
- **Distribuer les questions à plusieurs personnes pour améliorer la qualité de l'évaluation.**
- **Envisager de rendre les entretiens anonymes pour garantir l'honnêteté.**
- **Ne pas prendre les questions trop littéralement.**

6.3 Définition de la cible souhaitée

6.3.1 Déroulement

La définition de la cible souhaitée consiste à l'élaboration d'un score cible pouvant être utilisé comme instrument de mesure pour guider une organisation pour choisir les activités les plus importantes pour améliorer son score de maturité.

A cette étape, l'évaluateur définit ou actualise la cible en identifiant les activités que l'organisation devrait idéalement mettre en œuvre. En règle générale, cela comprendra plus d'activités de niveau inférieur que d'activités de niveau supérieur. Il faudrait également estimer l'impact de la cible choisie sur l'organisation.

6.3.2 Recommandations

- **Tenir compte du profil de risque de l'organisation.**
- **Respecter les dépendances entre les activités.**
- **S'assurer que l'ensemble total des activités sélectionnées a du sens.**
- **À titre de mesure approximative, l'impact global d'un effort d'assurance logicielle est estimé entre 5% et 10% du coût total de développement.**
- **Tirer parti de la feuille de calcul de la feuille de route dans la boîte à outils SAMM pour calculer les améliorations du score de maturité en fonction des réponses futures.**

6.4 Définition du plan

6.4.1 Déroulement

Cette étape a pour objectif de développer et mettre à jour le plan d'une organisation pour atteindre la cible souhaitée, améliorer son score et passer à un niveau supérieur de maturité.

6.4.2 Recommandations

- **Choisir une stratégie de changement réaliste en termes de nombre et de durée des phases. Une feuille de route typique se compose de 4 à 6 phases pendant 3 à 12 mois.**
- **Identifier les activités qui peuvent être réalisées rapidement et avec succès.**
- **Répartir la mise en œuvre des activités supplémentaires sur les différentes phases de la feuille de route, en tenant compte de l'effort requis pour les mettre en œuvre.**
- **Essayer d'équilibrer l'effort de mise en œuvre sur les différentes périodes et prendre en compte les dépendances entre les activités.**
- **Veiller sur la sensibilisation / la formation.**
- **S'adapter aux prochains cycles de publication / projets clés.**

6.5 Mise en place

6.5.1 Déroulement

Cette étape consiste à mettre en place toutes les actions nécessaires au déroulement du plan que l'organisation a validé pour passer à un niveau supérieur de maturité. Il faudrait tenir compte de l'impact de chaque action sur les processus, les personnes, les connaissances et les outils de l'organisation. Le présent document contient des conseils normatifs sur la façon de procéder.

6.5.2 Recommandations

- **Traiter les anciens logiciels séparément.**
- **N'imposer pas la migration à moins qu'elle ne soit vraiment indispensable.**
- **Éviter les goulots d'étranglement opérationnels, en particulier pour l'équipe de sécurité.**

6.6 Mise à disposition

6.6.1 Déroulement

Cette dernière étape a pour objectif principal de s'assurer que les améliorations sont disponibles et utilisées efficacement au sein de l'organisation et mesurer l'adoption et l'efficacité des améliorations mises en œuvre en analysant leur utilisation et leur impact.

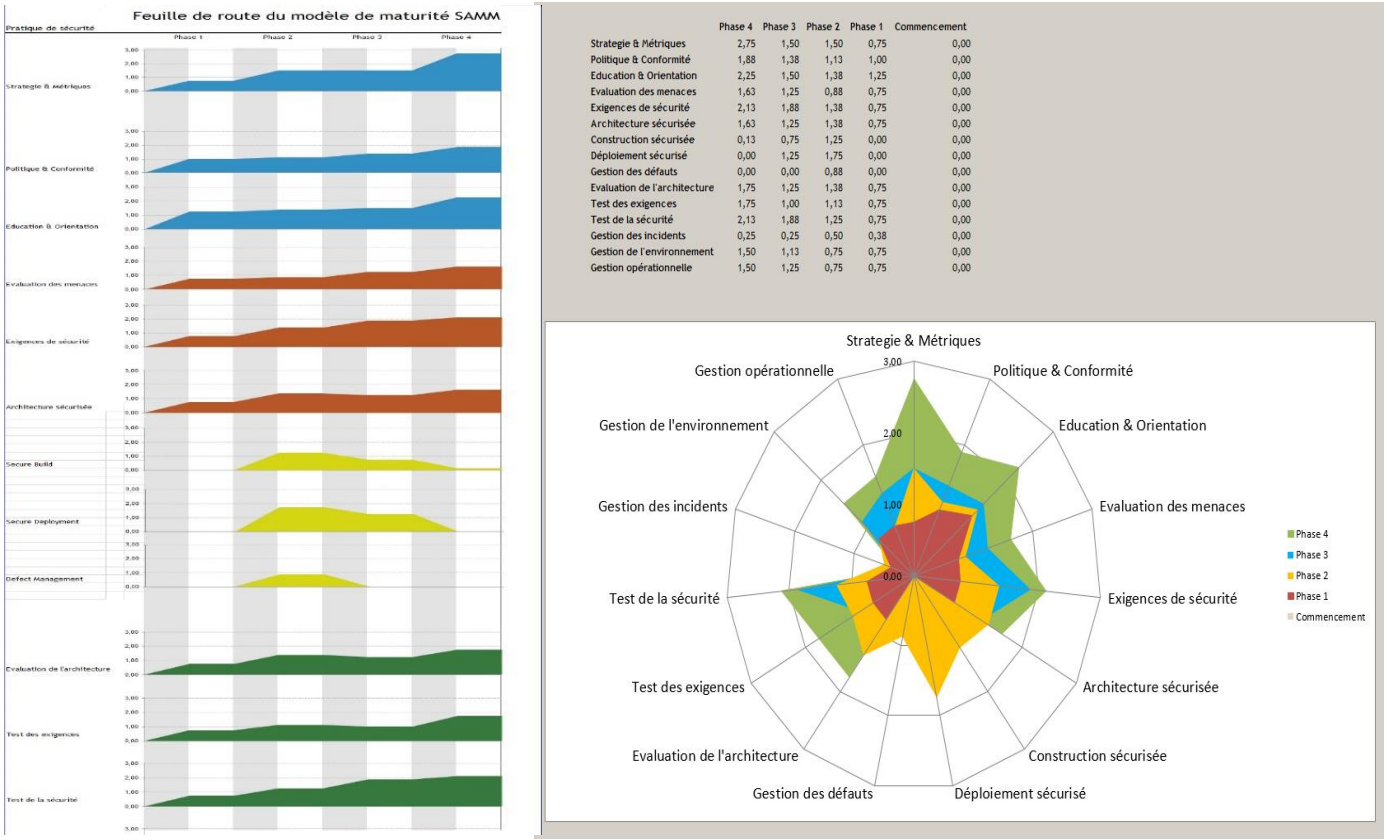
6.6.2 Recommandations

- **Rendre les étapes et les améliorations visibles pour toutes les personnes impliquées en organisant des formations et en communiquant avec les parties prenantes de la direction.**
- **Catégoriser les applications en fonction de leur impact sur l'organisation.**
- **Se Concentrer sur les applications à fort impact.**
- **Utiliser des champions d'équipe pour diffuser de nouvelles activités dans toute l'organisation.**

6.7 Exemple de la simulation de la matrice

La première étape de la mise en œuvre d'une approche d'amélioration continue de la sécurité des logiciels d'une entreprise consiste à évaluer sa situation actuelle. La matrice de calcul du niveau de la maturité est très recommandée pour toutes les parties prenantes, et ce, quel que soit leurs domaines, les technologies et les méthodologies de gestion des cycles de vie de leurs applications. Elle est **open et disponible** sur le web de la DGSSI. Elle offre un questionnaire détaillant toutes les fonctions métier, les pratiques de sécurité et les flux, et surtout permet à tout moment de connaitre le score de la partie prenante et son évolution dans le temps.

Les prises d'écran ci-après présentent les tableaux et les diagrammes relatifs à une simulation de résultat suite à l'utilisation de la matrice de calcul du niveau de la maturité.



	Phase 4	Phase 3	Phase 2	Phase 1	Commencement
Strategie & Métriques	2,75	1,50	1,50	0,75	0,00
Politique & Conformité	1,88	1,38	1,13	1,00	0,00
Education & Orientation	2,25	1,50	1,38	1,25	0,00
Evaluation des menaces	1,63	1,25	0,88	0,75	0,00
Exigences de sécurité	2,13	1,88	1,38	0,75	0,00
Architecture sécurisée	1,63	1,25	1,38	0,75	0,00
Construction sécurisée	0,13	0,75	1,25	0,00	0,00
Déploiement sécurisé	0,00	1,25	1,75	0,00	0,00
Gestion des défauts	0,00	0,00	0,88	0,00	0,00
Evaluation de l'architecture	1,75	1,25	1,38	0,75	0,00
Test des exigences	1,75	1,00	1,13	0,75	0,00
Test de la sécurité	2,13	1,88	1,25	0,75	0,00
Gestion des incidents	0,25	0,25	0,50	0,38	0,00
Gestion de l'environnement	1,50	1,13	0,75	0,75	0,00
Gestion opérationnelle	1,50	1,25	0,75	0,75	0,00

Phase 2 - Score de maturité

Fonctions métiers	Pratiques de sécurité	Actuel	Maturity 1	2	3
Gouvernance	Stratégie & Métriques	1,50	0,50	0,50	0,50
Gouvernance	Politique & Conformité	1,13	0,50	0,38	0,25
Gouvernance	Education & Orientation	1,38	0,75	0,38	0,25
Conception	Evaluation des menaces	0,88	0,25	0,38	0,25
Conception	Exigences de sécurité	1,38	0,38	0,38	0,63
Conception	Architecture sécurisée	1,38	0,63	0,38	0,38
Implémentation	Construction sécurisée	1,25	0,63	0,38	0,25
Implémentation	Déploiement sécurisé	1,75	0,50	0,50	0,75
Implémentation	Gestion des défauts	0,88	0,13	0,25	0,50
Vérification	Evaluation de l'architecture	1,38	0,50	0,50	0,38
Vérification	Test des exigences	1,13	0,25	0,50	0,38
Vérification	Test de la sécurité	1,25	0,50	0,38	0,38
Opérations	Gestion des incidents	0,50	0,13	0,25	0,13
Opérations	Gestion de l'environnement	0,75	0,25	0,25	0,25
Opérations	Gestion opérationnelle	0,75	0,25	0,25	0,25

Fonctions métiers	Actuel
Gouvernance	1,33
Conception	1,21
Implémentation	1,29
Vérification	1,25
Opérations	0,67

Phase 2 - Score de maturité

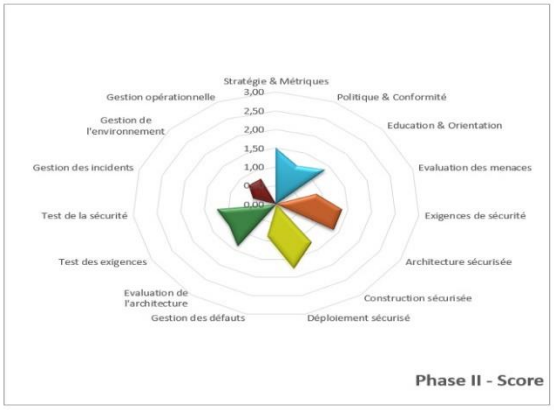


Figure 22 Matrice de calcul du niveau de la maturité

6.8 Synthèse

La meilleure façon d'appréhender SAMM est de commencer à l'utiliser. Ce chapitre a présenté un certain nombre d'étapes concrètes et du matériel de soutien pour vous aider à démarrer. La DGSSI vous invite à passer un jour ou deux à suivre les premières étapes, vous pourrez alors comprendre et apprécier la valeur ajoutée du modèle.

7 Bonnes pratiques SDLC

7.1 Correction des logiciels et des systèmes

De nombreux attaquants exploitent des vulnérabilités connues associées à des logiciels anciens ou obsolètes. Pour contrecarrer les attaques courantes, il faut toujours s'assurer que tous les systèmes disposent de correctifs à jour. L'application de correctifs réguliers est l'une des pratiques de sécurité logicielle les plus efficaces.

De nos jours, plus que la moitié des composants logiciels des applications sont open source. Pour ce, il faut veiller au maintien d'un inventaire, ou d'une nomenclature logicielle BOM[30], de ces composants. Cette nomenclature aide à s'assurer que les obligations liées aux licences de ces composants sont toutes respectées et que les correctifs liés à chaque composant sont communiqués.

Il est difficile de créer manuellement une nomenclature logicielle. Pour ce, la DGSSI recommande l'utilisation d'un outil d'analyse de la composition logicielle SCA[31] qui automatisera la tâche et mettra en évidence les risques de sécurité et de licence.

7.2 Formation des utilisateurs

La formation des employés doit faire partie de l'ADN[32] de sécurité de toutes les organisations : Avoir un programme de formation à la sécurité bien organisé et bien entretenu pour les employés contribuera grandement à protéger aussi bien les données que les actifs.

Il est recommandé d'inclure une formation de sensibilisation pour tous les employés et une formation sur le codage sécurisé pour les développeurs. La fréquence de ces formations doit être régulière, pas seulement une fois par an, avec réalisation des simulations comme des tests de phishing pour aider les employés à repérer et à arrêter les attaques d'ingénierie.

7.3 Automatisation des tâches de routine

Les attaquants utilisent l'automatisation pour détecter les ports ouverts, les erreurs de configuration de la sécurité, etc. Donc, défendre un système en utilisant uniquement des techniques manuelles n'est pas suffisant. Au lieu de cela, il est recommandé d'automatiser les tâches de sécurité quotidiennes, telles que l'analyse des modifications du pare-feu et des configurations de sécurité des appareils. L'automatisation des tâches fréquentes permet au personnel de sécurité de se concentrer sur des initiatives de sécurité plus stratégiques.

7.4 Application du moindre privilège

La DGSSI recommande que les utilisateurs et les systèmes disposent des privilèges d'accès minimum requis pour exécuter leurs fonctions. L'application du principe du moindre privilège réduit considérablement la surface d'attaque en éliminant les droits d'accès inutiles, ce qui peut entraîner une variété de compromis.

Cela implique d'éviter le glissement de privilèges, qui se produit lorsque les administrateurs ne révoquent pas l'accès aux systèmes ou aux ressources dont un employé n'a plus besoin. Une dérive des privilèges peut se produire lorsqu'un employé passe à un nouveau rôle, adopte de nouveaux processus, quitte l'organisation ou aurait dû recevoir un accès temporaire ou de niveau inférieur en premier lieu.

7.5 Création d'un plan de réponse aux incidents solide

Peu importe à quel point une organisation adhère aux bonnes pratiques de sécurité logicielle, elle sera toujours confrontée à la possibilité d'une violation. Mais si elle se prépare, elle peut empêcher les attaquants d'accomplir leur mission même s'ils violent ses systèmes.

Pour ce, la DGSSI recommande la mise en place d'un plan de réponse aux incidents IR[33] solide pour détecter une attaque, puis en limiter les dégâts.

7.6 Documentation des politiques de sécurité

Maintenir un référentiel de connaissances qui comprend des politiques de sécurité logicielles documentées de manière complète. Les politiques de sécurité permettent aux employés, y compris les administrateurs réseau, le personnel de sécurité, etc., de comprendre quelles activités leur organisation effectue et pourquoi.

De plus, il ne suffit pas d'avoir des politiques. Il faut s'assurer que tout le monde les lit. Au minimum lors du processus d'intégration des nouveaux employés.

[30] BOM: Bill of Materiel
[31] SCA: Software Composition Analysis
[32] AND: Deoxyribonucleic Acid
[33] IR: Incident Response

7.7 Segmentation du réseau

Segmenter le réseau est une application du principe du moindre privilège. Une bonne segmentation du réseau limite le mouvement des attaquants.

La DGSSI vous recommande d'identifier l'emplacement de stockage de toutes les données critiques et utiliser les contrôles de sécurité appropriés pour limiter le trafic à destination et en provenance de ces segments de réseau.

7.8 Intégration de la sécurité dans le cycle de vie de développement logiciel

Il faut veiller à intégrer les activités de sécurité logicielle dans le cycle de vie de développement logiciel SDLC du début à la fin. Ces activités doivent inclure :

- **L'analyse des risques de l'architecture ;**
- **Les tests de la sécurité des applications statiques, dynamiques et interactifs ;**
- **La SCA**[34] **;**
- **Et les tests de stylet.**

L'intégration de la sécurité dans le cycle de vie des développements logiciels SDLC nécessite d'abord du temps et des efforts. Mais corriger les vulnérabilités au début du SDLC est à la fois moins cher et plus rapide que d'attendre la fin.

7.9 Logging et monitoring

Le logging et le monitoring des événements sont vitaux pour la gestion de la sécurité. Ils doivent permettre de détecter au plus tôt les intrusions, les actions anormales et aussi de corriger les suites d'une attaque.

La journalisation est primordiale et il est fortement recommandé de journaliser les informations relatives à l'utilisation des données (comme le nombre et la taille des transactions) en plus des événements usuels.

L'utilisation d'un agrégateur de logs comme WEF[35] et d'un outil de monitoring comme Splunk ou la stack ELK[36] est vivement conseillée pour obtenir une vue d'ensemble de l'état du parc applicatif.

7.10 Définition des indicateurs clés

La DGSSI recommande la définition des indicateurs clés qui sont significatifs et pertinents pour chaque organisation : Les mesures bien définies permettent d'évaluer la posture de la sécurité au fil du temps.

7.11 Sensibilisation, Education et Formation

Parallèlement à toutes ces bonnes pratiques qui interviennent séquentiellement durant le cycle de développement, il est important de sensibiliser les équipes par le biais de l'AET[37] (Awareness, Education and Training). La formation aux principes de base du Secure Coding reste en effet un excellent moyen de prévention.

Une attention particulière devrait être accordée à la validation des entrées, à l'encodage des sorties, à l'authentification, à l'autorisation, à la gestion des sessions, aux contrôles d'accès, aux bonnes pratiques cryptographiques, à la gestion des erreurs et à la journalisation des événements. Il est également judicieux de prendre le temps d'aborder les problématiques de la désérialisation et de la compromission de librairies tierces, lesquelles sont aujourd'hui devenues un risque non négligeable pour les développements internes, et donc d'aborder les thématiques de Sub Resources Integrity et de gestion des dépendances.

7.12 Synthèse

Implémenter l'ensemble de ces bonnes pratiques n'est pas simple et nécessite du temps ainsi que des ressources spécialisées. Il n'est donc pas rare de devoir faire des choix, ne serait-ce qu'en termes de priorisation. Il est dans ce cas vivement recommandé de commencer par implémenter les processus qui permettront de supprimer un maximum de vulnérabilités.

En termes de sécurité applicative, les mesures les plus efficaces sont à prioriser dans cet ordre :

- **De procéder à tests unitaires au niveau de l'équipe de développement, afin d'obtenir un premier niveau de vérification que les différentes fonctionnalités ne sont bien accessibles que dans le contexte initialement prévu [par exemple en testant l'accès à une API après suppression du token associé].**
- **L'analyse automatisée du code source (avec une solution comme Checkmarx OSA**[38]**.**
- **L'analyse itérative des risques relatifs à l'architecture.**
- **Des tests d'intrusion applicatifs, en mettant l'application au cœur de l'analyse.**
- **Des tests automatisés de sécurité au niveau de l'équipe qualité.**

[34] SCA: Software Composition Analysis
[35] WEF: Windows Event Forwarding
[36] ELK: Elasticsearch, Logstash, and Kibana
[37] AET: Awareness, Education and Training
[38] OSA: Open Source Analysis

- **L’extension des Uses Cases conventionnels aux Abuses Cases.**

Au niveau de la sécurité opérationnelle, il convient de suivre les bonnes pratiques du NIST et de prioriser également les mesures les plus efficaces dans cet ordre :

- **Filtrer les flux à l’aide de pares-feux réseaux et applicatifs. Il est important de ne pas exposer inutilement de services qui pourraient être utilisés à mauvais escient (comme les ports TCP 2375 et 2376 qui permettent d’accéder aux APIs de gestion de Docker).**
- **Implémenter des scans automatisés de vulnérabilité.**
- **Remonter les journaux d’événements sur un serveur centralisé.**
- **Maintenir à jour l’inventaire du matériel et des applications, en veillant particulièrement à savoir en tout temps ce qui est exposé sur Internet.**
- **Patcher les composants, les librairies et les middlewares.**
- **Renforcer les codes compilés à l’aide d’outils de reverse engineering.**
- **Chiffrer les flux applicatifs contre l’écoute réseau et le rejeu.**
- **Renforcer les mécanismes d’authentification et de gestion des habilitations.**

8 Cadre de développement sécurisé (Modèle NIST)

Le SSDF[39] de NIST est un cadre qui comble une lacune dans les approches actuelles de la sécurité dans le développement des logiciels. Si les recommandations du SAMM sont adoptées, les bases liées à la capture des exigences de la sécurité des systèmes et de la sécurité dans le processus de développement et de support sont couvertes : Le niveau de couverture est calculable via la matrice de calcul du niveau de la maturité.

Cependant, pour veiller à ce que les intervenants dans le cycle de développement disposent de processus et de listes de contrôle pour les bibliothèques sécurisées à utiliser, quand et comment réutiliser un code, avec une gestion rapide et efficace des vulnérabilités, le SSDF de NIST permet toute cette gestion via quatre groupes de pratiques qui abordent tous les aspects du développement logiciel où des vulnérabilités pourraient être introduites :

- **Préparer l'organisation (PO) ;**
- **Protéger le logiciel (PS) ;**
- **Produire des logiciels bien sécurisés (PW) ;**
- **Répondre aux vulnérabilités (RV).**

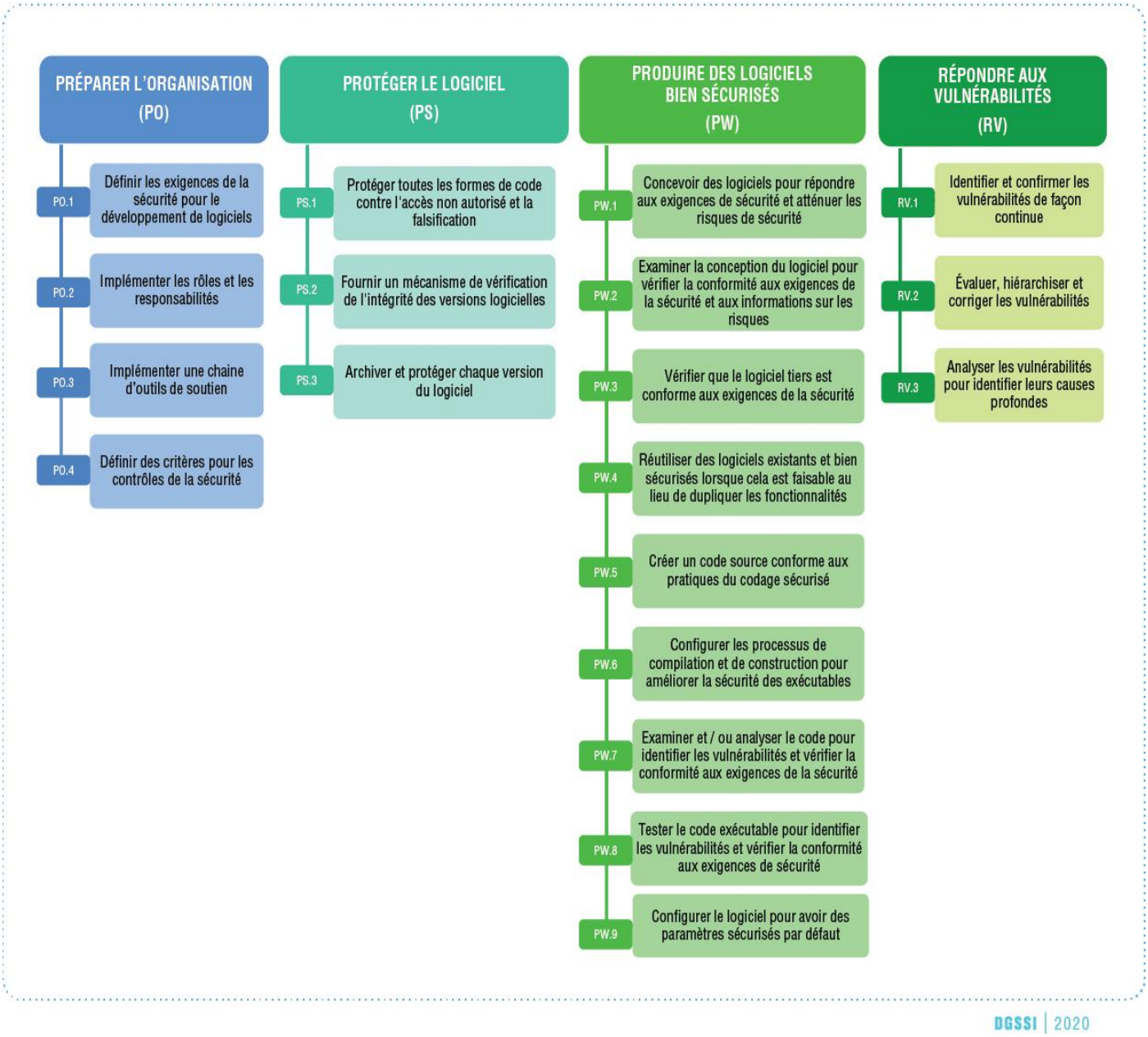


Figure 23 Modèle NIST

[39] SSDF: Secure Software Development Framework

8.1 Préparer l'organisation (PO)

Le groupe de pratiques **(PO)** prépare le terrain pour le développement d'un logiciel sécurisé. Il s'agit de noter toutes les exigences en matière de sécurité, d'identifier les personnes responsables, etc. Il s'agit également d'identifier les outils qui seront utilisés pendant tout le cycle de développement logiciel. Et enfin, cela inclut la liste des contrôles de sécurité que le logiciel devra passer.

8.1.1 Définir les exigences de la sécurité (PO.1)

L'équipe sécurité devra s'assurer que les exigences de sécurité pour le développement de logiciels sont connues à tout moment afin qu'elles puissent être prises en compte tout au long du SDLC et que la duplication des efforts puisse être minimisée car les informations sur les exigences peuvent être collectées une seule fois et partagées. Cela comprend les exigences provenant des sources internes (par exemple, les politiques de l'organisation, les objectifs commerciaux et la stratégie de gestion des risques) et de sources externes (par exemple, les lois et les réglementations applicables).

Tâche PO.1.1	Identifier toutes les exigences de sécurité applicables pour le développement logiciel général de l'organisation et les maintenir au fil de temps.

8.1.2 Implémenter les rôles et les responsabilités (PO.2)

L'organisation doit se préparer à un changement, et le changement doit être mis en œuvre aux plus hauts niveaux puisque l'adaptation touchera toutes les divisions. L'équipe de sécurité devra inclure des évaluateurs ayant une certaine expérience du développement et des opérations afin qu'ils puissent assister aux réunions, comprendre la terminologie et communiquer avec les équipes externes. L'équipe de sécurité doit avoir un plan pour mettre en œuvre les exigences et être en mesure de définir comment un logiciel sécurisé améliorera l'organisation. Cela nécessite des rôles dans le développement, la sécurité, les opérations, la gestion, les propriétaires de produit et tout autre rôle décisionnel. Une approche politique basée sur les exigences établies améliorera la posture de sécurité du code développé.

Tâche PO.2.1	- Créer de nouveaux rôles et modifier les responsabilités des rôles existants pour englober toutes les parties du SSDF. - Passer régulièrement en revue les rôles et les responsabilités définis et les mettre à jour au besoin.
Tâche PO.2.2	- Fournir une formation spécifique à un rôle pour tout le personnel ayant des responsabilités qui contribuent au développement sécurisé - Passer régulièrement en revue la formation spécifique au rôle, la mettre à jour au besoin
Tâche PO.2.3	Obtenir l'engagement de la haute direction pour sécuriser le développement et transmettre cet engagement à tout le personnel ayant des rôles et des responsabilités qui contribuent au développement sécurisé.

8.1.3 Implémenter une chaine d'outils de support (PO.3)

L'équipe de sécurité devra utiliser l'automatisation pour réduire l'effort humain nécessaire et améliorer l'exactitude, la cohérence et l'exhaustivité des pratiques de sécurité dans tout le SDLC. Elle devra également fournir un moyen pour documenter et démontrer l'utilisation de ces pratiques. Les chaînes d'outils et les outils peuvent être utilisés à différents niveaux de l'organisation, par exemple à l'échelle de l'organisation ou propres à un projet.

Tâche PO.3.1	Spécifier quels outils ou types d'outils doivent être inclus dans chaque chaine d'outils et lesquels sont obligatoires, ainsi que la manière dont les composants de la chaine d'outils doivent être intégrés les uns aux autres.
Tâche PO.3.2	En s'appuyant sur les bonnes pratiques de sécurité au niveau de tout le SDLC : - déployer et configurer les outils ; - les intégrer dans la chaine d'outils ; - maintenir les outils individuels et la chaine d'outils dans son ensemble.
Tâche PO.3.3	Configurer les outils pour collecter des preuves et des artefacts de leur prise en charge des pratiques du développement de logiciels sécurisés.

8.1.4 Définir des critères pour les contrôles de sécurité (PO.4)

Les vérifications des outils sélectionnés doivent permettre de produire des logiciels répondant aux critères de sécurité définis. La sortie des outils importés dans le pipeline d'orchestration doit être dans un format qui peut être utilisé pour déterminer les critères de réussite / échec pour chaque étape. Par exemple, l'organisation peut définir un nombre ou un niveau maximum de vulnérabilités produites par le test de sécurité d'application statique (SAST) ; si le niveau des résultats est dépassé, la génération échouera. Enfin, l'organisation doit définir des métriques pour mesurer l'impact des outils et du processus de sécurité sur le cycle de développement logiciel existant.

Tâche PO.4.1	Définir des critères pour les contrôles de sécurité des logiciels dans tout le SDLC.
Tâche PO.4.2	Mettre en œuvre des processus et des mécanismes pour rassembler les informations nécessaires à l'appui des critères

8.2 Protéger le logiciel (PS)

Le groupe de pratiques **(PS)** se concentre sur trois domaines principaux : la protection du code logiciel au sein de votre organisation, la garantie de l'intégrité des versions et l'archivage et la protection des versions.

8.2.1 Protéger toutes les formes de code contre l'accès non autorisé et la falsification (PS.1)

La pratique **(PS.1)** aide à empêcher les modifications non autorisées du code, intentionnelles et/ou accidentelles, qui pourraient contourner ou annuler les caractéristiques de la sécurité prévues du logiciel.

Tâche PS.1.1	Stocker toutes les formes de code, y compris le code source et le code exécutable, sur la base du principe du moindre privilège afin que seul le personnel autorisé dispose des formes d'accès nécessaires.

8.2.2 Fournir un mécanisme de vérification de l'intégrité des versions logicielles (PS.2)

La pratique **(PS.2)** aide les consommateurs de logiciels à s'assurer que le logiciel qu'ils acquièrent est légitime et n'a pas été falsifié.

Tâche PS.2.1	Mettre les informations de vérification à la disposition des consommateurs de logiciels.

8.2.3 Archiver et protéger chaque version du logiciel (PS.3)

La pratique **(PS.3)** aide à identifier, analyser et éliminer les vulnérabilités découvertes dans un logiciel après sa sortie.

Tâche PS.3.1	Archiver en toute sécurité une copie de chaque version et de tous ses composants (par exemple, le code, les fichiers de package, les bibliothèques tierces, la documentation) et les informations de vérification de l'intégrité de la version.

8.3 Produire des logiciels bien sécurisés (PW)

Le groupe de pratiques **(PW)** est destiné principalement aux intervenants dans le cycle de développement des logiciels qui devront se concentrer sur l'écriture du code en toute sécurité. C'est la plus grande section du NIST SSDF - à juste titre. Cela implique des domaines tels que la modélisation des menaces, l'utilisation de bibliothèques tierces sécurisées, les pratiques de codage sécurisées, la sécurité de construction et de compilation, les tests du code et le déploiement sécurisé.

Ces domaines peuvent avoir différentes implémentations au sein d'une organisation. Cela dépend du type d'application ainsi que de son architecture, du cadre choisi, etc. - sans parler des exigences de sécurité de l'entreprise.

8.3.1 Concevoir des logiciels pour répondre aux exigences de sécurité et atténuer les risques de sécurité (PW.1)

L'équipe sécurité devra identifier et évaluer les exigences de sécurité applicables pour la conception du logiciel ; déterminer les risques de sécurité auxquels le logiciel est susceptible de faire face pendant l'opération de production et trouver comment ces risques devraient être atténués par la conception du logiciel ; et justifier tous les cas où les décisions fondées sur les risques concluent que les exigences de sécurité devraient être assouplies ou supprimées. La prise en compte des exigences et des risques en matière de sécurité lors de la conception du logiciel (sécurisé par la conception) contribue à rendre le développement logiciel plus efficace.

Tâche PW.1.1	Utiliser des formes de modélisation des risques, telles que la modélisation des menaces, la modélisation des attaques ou la cartographie de la surface d'attaque, pour aider à évaluer les risques de la sécurité du logiciel.

8.3.2 Examiner la conception du logiciel pour vérifier la conformité aux exigences de sécurité et aux informations sur les risques (PW.2)

La pratique **(PW.2)** a pour but principal de s'assurer que le logiciel répond aux exigences de la sécurité et traite de manière satisfaisante les informations sur les risques identifiés. Et ce, en passant en revue la conception du logiciel pour confirmer qu'elle répond à toutes les exigences de sécurité, en examinant les modèles de risque créés lors de la conception du logiciel pour déterminer s'ils identifient correctement les risques, en examinant la conception du logiciel pour confirmer qu'elle répond de manière satisfaisante aux risques identifiés par les modèles de risque.

Tâche PW.2.1	Demander à une personne qualifiée qui n'a pas participé à la conception du logiciel de l'examiner pour confirmer qu'il réponde à toutes les exigences de sécurité et traite de manière satisfaisante les informations sur les risques identifiés.

8.3.3 Vérifier que le logiciel tiers est conforme aux exigences de sécurité (PW.3)

La pratique **(PW.3)** vise la réduction du risque associé à l'utilisation des modules et des services logiciels acquis, qui sont des sources potentielles de vulnérabilités supplémentaires.

Les objectifs de cette pratique peuvent être atteints via la mise en place de plusieurs actions dont :

- **La définition d'un ensemble d'exigences de sécurité de base et les inclure dans les documents d'acquisition, les contrats logiciels et tout autre accord avec des tiers ;**
- **La définition des critères liés à la sécurité pour la sélection des logiciels commerciaux et open source ;**
- **L'exigence des fournisseurs de modules et de services logiciels commerciaux qu'ils fournissent la preuve que leur logiciel est conforme aux exigences de sécurité de l'organisation ;**
- **L'établissement et le suivi des procédures pour faire face aux risques potentiels ;**
- **S'assurer que chaque module ou service logiciel est toujours activement maintenu ;**
- **Déterminer un plan d'action pour chaque module logiciel ou service tiers qui n'est plus maintenu ou disponible à l'avenir.**

Tâche PW.3.1	Communiquer les exigences aux tiers qui peuvent vous fournir des modules logiciels et des services en vue de leur réutilisation par vos logiciels.
Tâche PW.3.2	Utiliser des moyens appropriés pour vérifier que les modules et les services logiciels commerciaux et/ou open source et tous les autres services tiers sont conformes aux exigences.

8.3.4 Réutiliser des logiciels existants et bien sécurisés lorsque cela est faisable au lieu de dupliquer les fonctionnalités (PW.4)

L'adoption de la pratique **(PW.4)** réduit les coûts du développement des logiciels en les accélérant et en diminuant la probabilité d'introduire des vulnérabilités de sécurité supplémentaires dans le logiciel. Cela est particulièrement vrai pour les logiciels qui implémentent des fonctionnalités de sécurité, telles que les modules et protocoles cryptographiques.

Tâche PW.4.1	Acquérir des composants bien sécurisés (par exemple, des bibliothèques de logiciels, des modules, des intergiciels (middlewares), des Framework) auprès de tiers pour les utiliser par les logiciels de l'organisation.
Tâche PW.4.2	Créer des composants logiciels bien sécurisés en interne en suivant les processus SDLC pour répondre aux besoins de développement de logiciels internes communs qui ne peuvent pas être mieux satisfaits par des tiers.
Tâche PW.4.3	Le cas échéant, intégrer la prise en charge de l'utilisation des fonctionnalités et des services de sécurité normalisés (par exemple, intégration avec des systèmes de gestion des journaux, de gestion des identités, de contrôle d'accès et de gestion des vulnérabilités) au lieu de créer des implémentations propriétaires de fonctionnalités et de services de sécurité.

8.3.5 Créer un code source conforme aux pratiques de codage sécurisé (PW.5)

L'adoption de la pratique **(PW.5)** diminue le nombre des vulnérabilités de sécurité dans le logiciel et réduit les coûts en éliminant les vulnérabilités lors de la création du code source. Les objectifs de cette pratique peuvent être atteints via la mise en place de plusieurs actions dont la validation de toutes les entrées, la validation et l'encodage de toutes les sorties, l'utilisation des environnements de développement avec des fonctionnalités qui nécessitent l'utilisation de pratiques de codage sécurisées, etc.

Tâche PW.5.1	Suivre toutes les pratiques de codage sécurisées adaptées aux langages et à l'environnement de développement.
Tâche PW.5.2	Demander au développeur de réviser son propre code lisible par l'homme, d'analyser son propre code lisible par l'homme et / ou de tester son propre code exécutable pour compléter (et non remplacer) l'examen, l'analyse et / ou les tests de code effectués par d'autres.

8.3.6 Configurer les processus de compilation et de construction pour améliorer la sécurité des exécutables (PW.6)

La pratique **(PW.6)** diminue le nombre de vulnérabilités liées à la sécurité dans les logiciels et réduit les coûts en éliminant les vulnérabilités avant le test, en encourageant l'utilisation des versions à jour du compilateur et des outils de construction et en activant les fonctionnalités du compilateur qui produisent des avertissements pour le code mal sécurisé pendant le processus de compilation.

Tâche PW.6.1	Utiliser le compilateur et créez des outils qui offrent des fonctionnalités pour améliorer la sécurité des exécutables.
Tâche PW.6.2	Déterminer quelles fonctionnalités du compilateur et de l'outil de construction doivent être utilisées et comment chacune doit être configurée, puis implémenter la configuration approuvée pour les outils de compilation et de construction, les processus, etc.

8.3.7 Examiner et / ou analyser le code lisible par l'homme pour identifier les vulnérabilités et vérifier la conformité aux exigences de sécurité (PW.7)

L'adoption de la pratique **(PW.7)** aide à identifier les vulnérabilités afin qu'elles puissent être corrigées avant la sortie du logiciel pour éviter toute exploitation. L'utilisation de méthodes automatisées réduit l'effort et les ressources nécessaires pour détecter les vulnérabilités. Le code lisible par l'homme comprend le code source et toute autre forme de code qu'une organisation considère comme lisible par l'homme.

Tâche PW.7.1	Déterminer si la révision du code (c'est-à-dire, une personne regarde directement le code pour trouver des problèmes) et / ou l'analyse du code (c'est-à-dire, des outils sont utilisés pour trouver des problèmes dans le code, soit de manière entièrement automatisée ou en collaboration avec une personne) utilisé.
Tâche PW.7.2	- Effectuer la révision du code et / ou l'analyse du code en fonction des normes de codage sécurisé de l'organisation - Documenter et trier tous les problèmes découverts et les solutions recommandées dans le flux de travail ou le système de suivi des problèmes de l'équipe de développement.

8.3.8 Tester le code exécutable pour identifier les vulnérabilités et vérifier la conformité aux exigences de sécurité (PW.8)

La pratique **(PW.8)** aide à identifier les vulnérabilités pour qu'elles puissent être corrigées avant la sortie du logiciel afin d'éviter toute tentative malveillante de leur exploitation. L'utilisation des méthodes automatisées réduit l'effort et les ressources nécessaires pour détecter les vulnérabilités. Le code exécutable comprend les binaires, le byte code directement exécuté, le code source directement exécuté et toute autre forme de code qu'une organisation considère comme exécutable.

Tâche PW.8.1	Déterminer si le test du code exécutable doit être effectué et, le cas échéant, quels types de codes exécutables doivent être utilisés.
Tâche PW.8.2	- Concevoir et effectuer les tests - Documentez les résultats.

8.3.9 Configurer le logiciel pour avoir des paramètres sécurisés par défaut (PW.9)

L'adoption de la pratique **(PW.9)** améliore la sécurité du logiciel au moment de son installation pour réduire la probabilité qu'il soit déployé avec des paramètres de sécurité faibles qui l'exposeraient à un plus grand risque de compromission.

Tâche PW.9.1	Déterminer comment configurer chaque paramètre qui a un effet sur la sécurité afin que les paramètres par défaut soient sécurisés et n'affaiblissent pas les fonctions de sécurité fournies par la plate-forme, l'infrastructure réseau ou les services.
Tâche PW.9.2	Implémenter les paramètres par défaut (ou des groupes de paramètres par défaut, le cas échéant) et documenter chaque paramètre pour les administrateurs de logiciels.

8.4 Répondre aux rapports de vulnérabilités (RV)

Le domaine **(RV)** se concentre sur le post-déploiement. Son objectif principal est d'identifier les vulnérabilités sur une base continue, de les corriger et d'éliminer leur cause profonde. Cela peut être à l'échelle de l'organisation ou spécifique à une application. Cependant, il est logique d'avoir un système qui suit les bogues et les vulnérabilités.

8.4.1 Identifiez et confirmez les vulnérabilités de manière continue (RV.1)

La pratique **(RV.1)** a pour objectif principal que toutes les vulnérabilités soient identifiées plus rapidement afin qu'elles puissent être vite corrigées, ce qui réduit la fenêtre d'opportunité pour les attaquants.

Tâche RV.1.1	Rassembler des informations auprès des consommateurs et des sources publiques sur les vulnérabilités potentielles du logiciel et de tout composant tiers utilisé par le logiciel, et examiner tous les rapports crédibles.
Tâche RV.1.2	Passer en revue, analyser et / ou tester le code du logiciel pour identifier ou confirmer la présence de vulnérabilités non détectées auparavant.
Tâche RV.1.3	Avoir une équipe et un processus en place pour gérer les réponses aux rapports de vulnérabilité et aux incidents.

8.4.2 Évaluer, hiérarchiser et corriger les vulnérabilités (RV.2)

La pratique **(RV.2)** vise à s'assurer que les vulnérabilités sont corrigées aussi rapidement que nécessaire, ce qui réduit la fenêtre d'opportunité pour les attaquants.

Tâche RV.2.1	Analyser chaque vulnérabilité pour collecter suffisamment d'informations pour planifier sa correction.
Tâche RV.2.2	Développer et mettre en œuvre un plan de correction pour chaque vulnérabilité.

8.4.3 Analyser les vulnérabilités pour identifier leurs causes profondes (RV.3)

L'adoption de la pratique **(RV.3)** aide à réduire la fréquence des vulnérabilités à l'avenir.

Tâche RV.3.1	Analyser toutes les vulnérabilités identifiées pour déterminer la cause première de chaque vulnérabilité.
Tâche RV.3.2	Analyser les causes profondes au fil du temps pour identifier des modèles, par exemple lorsqu'une pratique de codage sécurisé particulière n'est pas suivie de manière cohérente.

8.5 Synthèse

Adopter les pratiques de la sécurité proposées par le modèle de NIST permet d'atteindre un niveau de développements des logiciels sécurisés idoine. Implémenter l'ensemble des bonnes pratiques de ce modèle n'est pas simple et nécessite du temps et des ressources spécialisées. Il est possible d'atteindre un bon niveau de maturité en agissant progressivement selon les priorités résultantes de l'étude de risque.

En effet, il est vivement recommandé de commencer par implémenter les processus qui permettront de supprimer un maximum de vulnérabilités. Les Frameworks SAMM et OWASP pourraient être un bon point de départ, il est possible de les mettre en place au niveau d'un petit périmètre et d'élargir progressivement le scope pour améliorer le niveau de maturité en adoptant une combinaison de plusieurs Frameworks pour répondre aux différentes pratiques de sécurité selon le modèle présenté dans la figure 24.

	BSA	BSIMM	IDASOAR	ISO 27034	MSSDL	OWASP	SAMM	SCAGILE
PO.1.1	X	X		X	X	X	X	
PO.2.1	X	X						
PO.2.2	X	X			X		X	X
PO.2.3		X					X	
PO.3.1	X				X			X
PO.3.2	X							X
PO.3.3	X				X			X
PO.4.1	X	X		X	X	X	X	
PO.4.2	X	X						
PS.1.1	X		X			X		
PS.2.1	X	X					X	
PS.3.1	X		X					
PW.1.1	X	X	X	X	X	X	X	X
PW.2.1	X	X		X		X	X	
PW.3.1	X	X	X		X		X	
PW.3.2	X		X		X	X		X
PW.4.1	X		X		X		X	
PW.4.2		X	X			X		
PW.4.3	X				X	X		
PW.5.1	X		X	X	X	X		
PW.6.1	X				X			X
PW.6.2	X					X		X
PW.7.2	X	X	X	X	X	X	X	X
PW.8.1	X							
PW.8.2	X	X	X	X	X		X	X
PW.9.1	X		X	X		X		X
PW.9.2			X			X		X
RV.1.1	X	X					X	X
RV.1.2	X			X				
RV.1.3	X				X		X	
RV.2.1	X							X
RV.2.2	X							X
RV.3.1	X						X	
RV.3.2	X							
RV.3.3	X							
RV.3.4	X	X			X			

Figure 24 Frameworks assurant les pratiques de sécurité du modèle NIST

9 Conclusion

Une application logicielle efficace est une application réalisée pour répondre à un besoin client et garantir une expérience utilisateur agréable dans un environnement totalement sécurisé. La question cruciale est de savoir comment un logiciel résiste aux failles de sécurité ? Les ingénieurs sont souvent moins conscients des approches de sécurité menant à des logiciels fonctionnels, mais très vulnérables aux menaces de sécurité à la fin du projet. En raison de diverses failles dans cette approche de développement, des problèmes de sécurité sont susceptibles de se produire lorsque l'accent est mis sur le rôle fonctionnel du logiciel, tandis que de nombreux problèmes de sécurité sont ignorés au cours du processus.

De nombreux ingénieurs tentent d'améliorer les aspects de sécurité en revisitant le travail une fois le processus de développement est terminé. Sans surprise, de nombreux produits logiciels soumis à des tests de sécurité sont jugés vulnérables aux menaces et ne fournissent pas un environnement sûr aux utilisateurs et aux clients.

Cela est probablement dû au manque de mesures systématiques telles que les examens, les procédures ou les cadres ; ces mesures peuvent aider les développeurs et les chefs de projet à s'assurer que les processus de sécurité sont suivis de manière cohérente tout au long du processus de développement, conformément à un ensemble de règles et de procédures. Par conséquent, une approche ou un cadre qui peut être utilisé dès le début d'un projet et suivi de manière cohérente tout au long est nécessaire pour la construction de mesures de sécurité.

Pour gérer efficacement les problèmes de sécurité qui existent dans de nombreux projets de développement, il est jugé nécessaire d'intégrer une réflexion axée sur la sécurité tout au long du processus de développement. Cela réduit le risque de louper des exigences de sécurité importantes ou de commettre des erreurs critiques dans la conception du logiciel.

En utilisant cette stratégie, les problèmes peuvent être découverts beaucoup plus tôt qu'à la fin lorsqu'ils sont très difficiles et coûteux à résoudre.

Les vulnérabilités associées au système peuvent être réduites à un niveau minimum lors du cycle de vie d’un développement logiciel SDLC. Il peut ne pas être possible d'éliminer la vulnérabilité, mais elle peut être réduite à un niveau minimum si la sécurité est traitée comme un processus continu.

Par conséquent, cette étude examine le processus de développement logiciel dans son ensemble, du point de vue de chaque phase de développement SDLC, et cherche à déterminer d'importantes mesures sécurisées qui doivent être utilisées à chaque phase pour garantir des produits hautement sécurisés.

10 Références

[1] Oracle Software Security Assurance (OSSA). Disponible sur https://www.oracle.com/corporate/security-practices/assurance/

[2] Processus logiciel d'équipe sécurisé (TSP-Secure). Disponible sur https://us-cert.cisa.gov/bsi/articles/knowledge/sdlc-process/secure-software-development-life-cycle-processes

[3] Microsoft Security Development Lifecycle (Microsoft SQDL). Disponible sur https://www.microsoft.com/en-us/sdl

[4] Building Security in Maturity Model (BSIMM) Version 10. Disponible sur https://www.bsimm.com/download/

[5] Open Web Application Security Project -- Software Assurance Maturity Model Version 2.0 (SAMM). Disponible sur https://owaspsamm.org/

[6] Comprehensive Lightweight Application Security Process (CLASP). Disponible sur https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf

[7] Une étude de cas. Disponible sur le https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9075225

[8] Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF). Disponible sur https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04232020.pdf

[9] Bonnes pratiques SDLC. Disponible sur https://www.synopsys.com/blogs/software-security/top-10-software-security-best-practices/

11 Liste des figures

Figure 1 Méthodologies de développement logiciel 7
Figure 2 Méthodologie Agile 7
Figure 3 Principes Lean 8
Figure 4 Modèle Cascade 8
Figure 5 Modèle Itératif 9
Figure 6 Modèle Spiral 10
Figure 7 Modèle DevOps 10
Figure 8 Etapes d'une Intégration Continue 12
Figure 9 Chaine de production 13
Figure 10 Cycle de vie d'une application logicielle moderne 14
Figure 11 Cycle de vie d'un logiciel selon une approche DevSecOps 15
Figure 12 Les cinq étapes du modèle SDL 16
Figure 13 Processus CLASP 17
Figure 14 Activités de sécurité BSIMM 20
Figure 15 Filtres de suppression des vulnérabilités 21
Figure 16 Fonctions métier SAMM 2.0 22
Figure 17 Niveaux de maturité SAMM 2.0 23
Figure 18 Comparaison des modèles étudiés 24
Figure 19 Flux géré par le modèle SAMM 2.0 26
Figure 20 Fonctions métier SAMM 2.0 27
Figure 21 Phases de la mise en place du SAMM 37
Figure 22 Matrice de calcul du niveau de la maturité 40
Figure 23 Modèle NIST 44
Figure 24 Frameworks assurant les pratiques de sécurité du modèle NIST 50

12 Liste des tableaux

Tableau 1 Pratiques BSIMM 19
Tableau 2 Activités de sécurité en fonction du SDLC 25
Tableau 3 Niveaux de maturité et notation pour le modèle SAMM 2.0 27
Tableau 4 Strategy & Metrics - Niveaux de maturité 28
Tableau 5 Policy & Compliance - Niveaux de maturité 28
Tableau 6 Education & Guidance – Niveaux de maturité 29
Tableau 7 Threat assessment – Niveaux de maturité 29
Tableau 8 Security requirements - Niveaux de maturité 30
Tableau 9 Secure architecture - Niveaux de maturité 31
Tableau 10 Secure build - Niveaux de maturité 31
Tableau 11 Secure Deployment - Niveaux de maturité 32
Tableau 12 Defect management - Niveaux de maturité 32
Tableau 13 Architecture assessment - Niveaux de maturité 33
Tableau 14 Requirements driven testing - Niveaux de maturité 34
Tableau 15 Security testing - Niveaux de maturité 34
Tableau 16 Incident management - Niveaux de maturité 35
Tableau 17 Environment management - Niveaux de maturité 35
Tableau 18 Operational Management - Niveaux de maturité 36

13 Acronymes

ADN	Deoxyribonucleic Acid
AET	Awareness, Education and Training
BOM	Bill of Materiel
BSIMM	Building Security in Maturity Model
CD	Continuous Delivery
CI	Continuous Integration
CIS	Computer information systems
CLASP	Comprehensive Lightweight Application Security Process
DAST	Dynamic Application Security Testing
DDoS	Distributed Denial of Service
Dev	Development
DevSecOps	Development Security Operations
DGSSI	Direction Générale de la Sécurité des Systèmes d'Information
EDR	Endpoint Detection and Response
EG	Education and Guidance
ELK	Elasticsearch, Logstash and Kibana
EM	Security Testing
HIDS	Host Intrusion Detection Systems
IAM	Identity and Access Management
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IR	Implementation Review
IT	Information Technology
Microsoft SDL	Microsoft Security Development Lifecycle
NIST	National Institute of Standards and Technology
Ops	Operations
OPSEC	Operations security
OSA	Open Source Analysis
OSSA	Oracle Software Security Assurance
OWASP	Open Web Application Security Project
PC	Policy and Compliance
QA	Quality assurance
RBAC	Role-Based Access Control
RUP	Rational Unified Process
SA	Secure Architecture
SAMM	Software Assurance Maturity Model
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SDL	Software Development Lifecycle
SDLC	Systems Development Life Cycle
Sec	Security
SEI	Software Engineering Institute
SM	Strategy and Metrics
SOC	Security Operations Center
SR	Security Requirements
SSG	Software Systems Group
ST	Security Testing
TA	Threat Assessment
TSP-Secure	Processus logiciel d'équipe sécurisé
WAF	Web Application Firewall
WEF	Windows Event Forwarding

Évaluation de la maturité de la sécurité du cycle de vie des développements logiciels

Guide de bonnes pratiques

UNE PUBLICATION DE LA

DIRECTION GENERALE DE LA SECURITE DES SYSTEMES D'INFORMATION

2021

contact@dgssi.gov.ma