УДК 004.415.2.056

Л. М. Куперштейн, канд. техн. наук, доц.; М. Б. Тарасюк;
В. І. Маліновський, канд. техн. наук, доц.

МЕТОДОЛОГІЧНІ ПІДХОДИ ДО БЕЗПЕЧНОЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В умовах стрімкого зростання кількості кіберзагроз та посилення нормативних вимог, забезпечення безпеки програмного забезпечення (ПЗ) стає критично важливим завданням на всіх етапах його життєвого циклу (ЖЦ). Стаття присвячена комплексному дослідженню сучасних підходів до організації життєвого циклу безпечної розробки програмного забезпечення (Secure Software Development Life Cycle – SSDLC). Концепція SSDLC передбачає проактивну інтеграцію принципів та практик кібербезпеки безпосередньо в процеси планування, проєктування, розробки, тестування та розгортання ПЗ, що дозволяє виявляти та усувати потенційні вразливості та загрози на ранніх стадіях, значно знижуючи ризики та вартість їх виправлення. У статті детально аналізуються шість провідних методологій, що реалізують принципи SSDLC, кожна з яких має свої унікальні особливості та сфери застосування, а саме Microsoft Security Development Life cycle (MSDL), Secure Software Development Framework (SSDF), Software Assurance Maturity Model (SAMM), Cisco Security Development Life cycle (CSDL), Oracle Software Security Assurance (OSSA) та Adobe Secure Product Life cycle (ASPL). Для кожної методології проведено порівняльний аналіз ключових характеристик, включаючи етапи життєвого циклу, основні переваги та недоліки. Особливу увагу приділено оцінці глибини та ефективності інтеграції практик безпеки на кожному етапі ЖЦПЗ, від визначення вимог до розгортання та підтримки. В статті представлено практичні рекомендацій щодо вибору найбільш прийнятного підходу SSDLC. Ці рекомендації базуються на врахуванні специфічних потреб організації, її розміру, наявних ресурсів, рівня зрілості наявних процесів розробки та безпеки, а також конкретних вимог до захищеності програмних продуктів. Підкреслюється, що впровадження SSDLC є не лише технічною необхідністю, але й стратегічною інвестицією в надійність, безпеку та конкурентоспроможність програмного забезпечення в сучасному цифровому світі.

Ключові слова: *життєвий цикл, розробка програмного забезпечення, інтеграція безпеки, кібербезпека, загроза, вразливість.*

Вступ

Сьогодні програмне забезпечення є критичним інструментом у всіх сферах діяльності, починаючи від фінансових і банківських послуг до управління інфраструктурою та комунікацій. Зростання кількості кіберзагроз та посилення нормативних вимог підвищує необхідність створення безпечного ПЗ. Інтеграція принципів кібербезпеки у процес розробки дає можливість компаніям не лише забезпечити захист своїх користувачів, але й дотримуватися сучасних стандартів безпеки. Пошук ефективних способів впровадження SSDLC актуальний для всіх, хто прагне створювати продукти високої надійності, захищені від потенційних атак та витоку даних.

Постановка задачі

Життєвий цикл безпечної розробки програмного забезпечення або SSDLC (Secure Software Development Life Cycle), є сучасним підходом, який передбачає інтеграцію безпеки на всіх етапах створення програмного забезпечення [1]. Метою SSDLC є створення застосунків, захищених від можливих загроз і потенційних вразливостей, що дозволяє знижувати ризики ще на ранніх етапах розробки.

Основна ідея SSDLC полягає у впровадженні безпеки в усі процеси розробки ПЗ, а саме: планування, проєктування, розробки, тестування та розгортання, що дозволяє уникати

додаткових витрат на доопрацювання та швидко реагувати на потенційні ризики [2].

На рис. 1. а та 1. б наведено схему порівняння етапів ЖЦ ПЗ та ЖЦ ПЗ з інтеграцією процесів забезпечення безпеки відповідно.

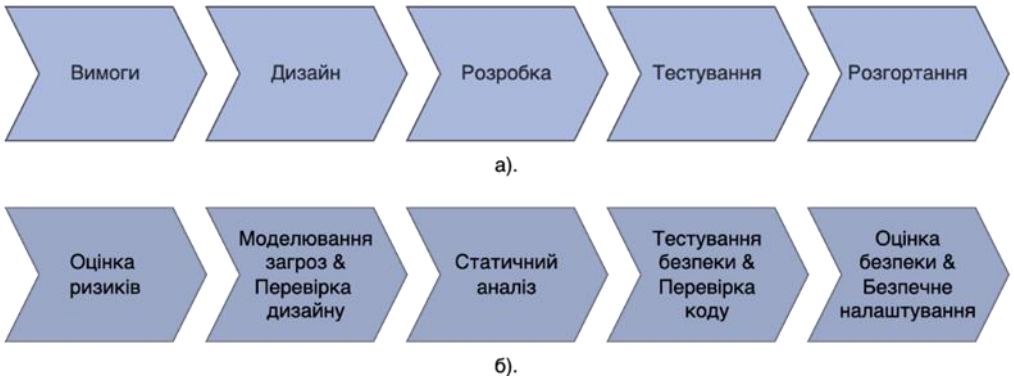

Рис. 1. Етапи методологій SDLC та SSDLC

SSDLC підхід акцентує увагу на виявленні та усуненні вразливостей до етапу розгортання продукту, забезпечуючи тим самим високий рівень захищеності і знижуючи ймовірність порушень безпеки після запуску [3].

Переваги SSDLC є значними і включають:

- підвищену безпеку кінцевого продукту завдяки впровадженню захисту на кожному етапі життєвого циклу;
- економію часу та ресурсів: виявлення і усунення вразливостей на ранніх етапах розробки є менш затратним, ніж виправлення їх у готовому продукті;
- підвищення довіри користувачів до продукту, оскільки він відповідає сучасним стандартам безпеки.

Попри очевидні переваги, існують перешкоди для впровадження SSDLC. Основними з них є необхідність додаткового навчання команди розробників, що потребує інвестицій, і можливий опір змінам з боку працівників, звиклих до традиційного процесу розробки [4]. Крім того, SSDLC може збільшувати час розробки, адже перевірка на безпеку відбувається на кожному етапі, що іноді зменшує гнучкість і швидкість процесів. Також збільшення тривалості розробки пов'язане безпосередньо і з збільшення фінансових витрат на розробку.

Метою статті є дослідження особливостей підходів до інтеграції безпеки в ЖЦ ПЗ для вибору більш прийнятного в залежності від потреб та рівня зрілості процесів компанії.

Результати дослідження

Загальна концепція SSDLC є основою для ряду методологій, кожна з яких по своєму реалізує процеси забезпечення безпеки. Розглянемо більш детальніше особливості різних імплементацій.

Microsoft Security Development Lifecycle (MSDL) – це модель безпечної розробки ПЗ, розроблена компанією Microsoft. Її мета – знизитикількість вразливостей у ПЗ через впровадження відповідних процесів безпеки. Життєвий цикл MSDL складається із таких етапів (рис. 2) [5]:

- Навчання. Передбачається, що розробники проходять навчання з питань безпеки перед початком проєкту.
- Визначення вимог. Тут відбувається включення вимог безпеки до загальних вимог до проєкту.
- Проєктування. Тут розробляється модель загроз та аналіз можливих ризиків безпеки.
- Реалізація. На цьому етапі передбачається використання інструментів для статичного аналізу коду і контроль за безпечним кодуванням.
- Тестування. На цьому етапі відбувається оцінювання якості безпеки через перевірку якості коду, тести на проникнення тощо.

– Впровадження. Тут передбачається інтеграція перевірки безпеки в процес деплойменту ПЗ.
– Реакція на інциденти. Тут передбачається моніторинг безпеки ПЗ після розгортання та підготовка до можливих інцидентів.

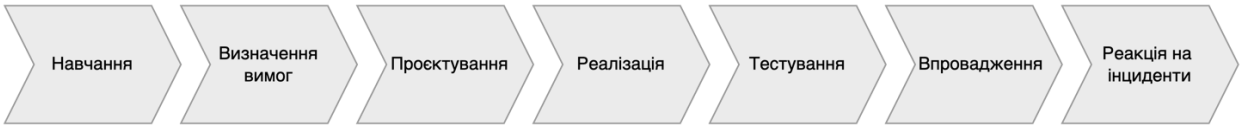

Рис. 2. Схема етапів MSDL

MSDL характеризується впровадженням початкової оцінки безпеки для кожного проєкту, що дозволяє ідентифікувати потенційні загрози на ранніх етапах розробки. Методологія передбачає чітке управління ризиками на основі загальноприйнятих принципів, а також безперервний аналіз вразливостей і впровадження політик безпеки [6].

Серед основних переваг MSDL слід відзначити високу ефективність в усуненні загроз завдяки стандартизації процесів безпеки, а також здатність методології ефективно адаптуватися до масштабних корпоративних проєктів. Водночас впровадження MSDL супроводжується значними витратами на навчання та інтеграцію, що є основним недоліком, поряд із тим, що методологія в основному орієнтована на продукти Microsoft.

MSDL доцільно застосовувати у великих організаціях, які прагнуть мінімізувати ризики безпеки та мають достатні ресурси для комплексної інтеграції цієї методології. Як приклад успішного використання MSDL можна розглянути впровадження цієї методології Microsoft у процесі розробки операційної системи Windows, що забезпечує високий рівень захисту на всіх етапах життєвого циклу продукту.

Таким чином, MSDL є прикладом стандартизації процесів безпеки в розробці програмного забезпечення, спрямованої на зменшення вразливостей і підвищення надійності продуктів.

Secure Software Development Framework (SSDF) – це підхід до розробки ПЗ, розроблений Національним інститутом стандартів і технологій США (NIST), який дозволяє інтегрувати безпеку у процес розробки ПЗ [7]. Основні принципи SSDF (рис. 3):

– Готувати організацію до безпечної розробки ПЗ включенням вимог безпеки у політику компанії та навчання персоналу.
– Інтегрувати безпеку на етапі дизайну за рахунок оцінки ризиків на етапі проєктування та врахування їх у архітектурі системи.
– Забезпечити безпеку під час розробки, а саме застосування інструментів для безпечного кодування, моніторинг змін у коді.
– Виявляти вразливості перед і після випуску. Це передбачає використання технік автоматизованого тестування для пошуку вразливостей та подальший моніторинг безпеки після випуску продукту.

На рис. 3 наведено схему фреймворку SSDF Національного інституту стандартів і технологій США.

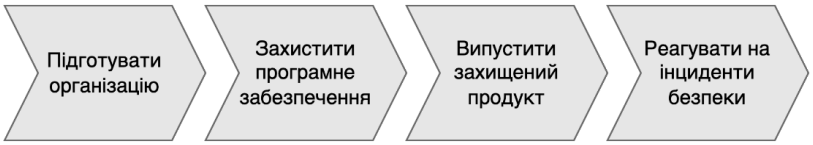

Рис. 3. Схема етапів SSDF

SSDF є методологією, що ґрунтується на загальноприйнятих стандартах безпеки та забезпечує ефективне управління ризиками на всіх етапах життєвого циклу розробки програмного забезпечення. Головною особливістю SSDF є інтеграція вимог безпеки, які узгоджуються з міжнародними нормами та стандартами, що дозволяє організаціям підтримувати високий рівень захищеності ПЗ [8].

Серед основних переваг SSDF варто виділити його відкритість і підтримку з боку

Національного інституту стандартів і технологій США (NIST). Завдяки модульній структурі цей фреймворк легко інтегрується в уже наявні процеси життєвого циклу безпечної розробки програмного забезпечення (SSDLC), що робить його зручним для компаній із різними рівнями зрілості безпекових процесів.

Проте, застосування SSDF потребує індивідуального налаштування відповідно до специфіки кожного проєкту, що може ускладнювати його впровадження. Крім того, ефективне використання фреймворку вимагає глибокого розуміння стандартів NIST, що може стати викликом для організацій, які не мають відповідного досвіду.

SSDF є оптимальним вибором для організацій, що працюють у регульованих галузях із підвищеними вимогами до безпеки, таких як державний та фінансовий сектори. Як приклад його застосування можна розглянути використання SSDF в урядових організаціях США для розробки програмного забезпечення, яке відповідає федеральним вимогам щодо безпеки.

Таким чином, SSDF сприяє розробці ПЗ, яке відповідає найкращим світовим практикам кібербезпеки, а також забезпечує безперервний контроль і підтримку безпеки після впровадження продукту.

Software Assurance Maturity Model (SAMM) – це модель зрілості, розроблена для оцінки та покращення процесів безпеки у розробці ПЗ [9]. Вона дозволяє організаціям визначити рівень безпеки в наявних процесах і поступово інтегрувати поліпшення для підвищення зрілості. SAMM складається з чотирьох основних доменів:

- Governance, що передбачає визначення політик безпеки та управління процесами безпеки.
- Design, що передбачає аналіз архітектури ПЗ та впровадження захисту на рівні проєктування.
- Implementation, що передбачає дотримання правил безпечного кодування, використання інструментів для аналізу коду.
- Verification, що передбачає забезпечення належного тестування безпеки продукту перед його випуском.
- Operations, що передбачає управління безпекою продукту після його впровадження, контроль за виправленням вразливостей.

На рис. 4 наведено схему моделі SAMM розробленої OWASP.

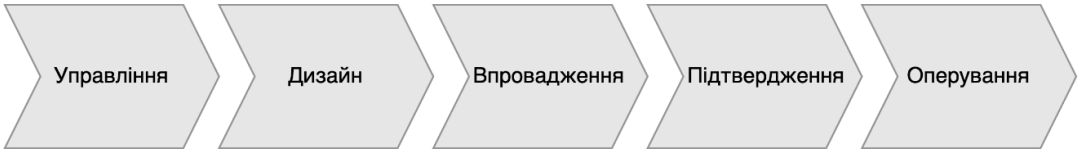

Рис. 4. Етапи методології SAMM

Software Assurance Maturity Model (SAMM) є гнучкою моделлю зрілості, яка дозволяє організаціям поступово впроваджувати заходи безпеки відповідно до їхніх потреб та поточного стану процесів. Модель охоплює ключові аспекти безпеки програмного забезпечення, включаючи стратегію безпеки, захист інфраструктури, управління ризиками та захист даних, що забезпечує комплексний підхід до побудови безпечної розробки [10].

До основних переваг SAMM належить здатність легко адаптуватися до будь-якого рівня зрілості безпеки в організації, що робить її придатною як для малих команд, так і для великих корпоративних структур. Модель передбачає поетапне впровадження заходів безпеки, що дозволяє мінімізувати витрати на старті та уникнути перевантаження процесів.

Проте впровадження SAMM вимагає постійного моніторингу та вдосконалення безпекових практик, оскільки модель орієнтована на поступове зростання зрілості, а не на досягнення фіксованого рівня відповідності. Це може потребувати додаткових зусиль з боку менеджменту та технічних фахівців.

Застосування SAMM рекомендовано організаціям, які прагнуть системно та поетапно покращувати свої процеси забезпечення безпеки, зокрема в динамічних середовищах, де

зміни відбуваються швидко, а вимоги до захисту ПЗ постійно зростають. Наприклад, модель може активно використовується у фінансових компаніях та великих корпораціях для довготривалої оптимізації процесів безпеки без порушення вже наявної структури.

Таким чином, SAMM дозволяє будувати адаптивну та керовану систему безпеки, яка відповідає поточним цілям організації та її рівню розвитку у сфері безпечної розробки.

Cisco Secure Development Lifecycle (CSDL) – методологія безпечної розробки від Cisco, яка інтегрує безпеку на всіх етапах створення продукту. Цей процес акцентує увагу на безпеці мережевих рішень та продуктів Cisco [11]. Виділяються такі етапи (рис. 5):

- Навчання. Розробники отримують спеціалізоване навчання з безпеки для обізнаності про нові загрози.
- Вимоги. Створення чітких вимог безпеки для кожного проєкту.
- Аналіз архітектури. Проведення моделювання загроз та аналізу архітектури на предмет вразливостей.
- Кодування. Використання безпечних практик кодування, відповідність внутрішнім стандартам Cisco.
- Перевірка. Валідація коду через статичний та динамічний аналіз, тестування на проникнення.
- Деплоймент. Впровадження продукту з урахуванням безпечного розгортання.
- Підтримка. Постійний моніторинг та оновлення для підтримки безпеки на тривалий час.

Рис. 5. Схема етапів CSDL

Методологія CSDL призначена для інтеграції безпеки у процес розробки програмного забезпечення, особливо в контексті мережевих рішень. Основними характеристиками CSDL є регулярне тестування компонентів програмного забезпечення на наявність вразливостей, а також забезпечення відповідності глобальним стандартам кібербезпеки. Такий підхід дозволяє підтримувати високий рівень захисту на всіх етапах створення мережевих продуктів [12].

Перевагою CSDL є його глибока орієнтація на безпеку продуктів мережевої інфраструктури, що дозволяє ефективно реагувати на загрози в середовищах з високими вимогами до надійності та стійкості. Методологія забезпечує багаторівневий захист, охоплюючи як сам процес розробки, так і після впроваджувальну підтримку безпеки.

Разом із тим, CSDL має певні обмеження у застосуванні, зокрема через свою орієнтацію переважно на продукти Cisco. Це робить методологію менш універсальною порівняно з іншими підходами, що можуть використовуватись у ширшому спектрі проєктів та технологій.

CSDL є оптимальним вибором для компаній, що розробляють або впроваджують мережеві рішення на базі Cisco і прагнуть досягти високого рівня безпеки інфраструктурних компонентів. Сама компанія Cisco активно застосовує CSDL у створенні власної мережевої інфраструктури, що дозволяє забезпечити надійний захист даних і стійкість до кіберзагроз.

Oracle Software Security Assurance (OSSA) – це методологія компанії Oracle, спрямована на інтеграцію заходів безпеки на всіх етапах життєвого циклу розробки програмного забезпечення. Основною метою OSSA є забезпечення того, щоб продукти Oracle допомагали клієнтам відповідати їхнім вимогам безпеки, забезпечуючи при цьому найбільш економічно ефективний досвід використання [13].

Основні характеристики OSSA [14]:

– Інтеграція безпеки на всіх етапах розробки. OSSA охоплює дизайн, розробку, тестування та обслуговування продуктів, забезпечуючи вбудовану безпеку на кожному етапі життєвого циклу.

– Використання стандартів безпечного кодування. Розробники дотримуються формальних стандартів безпечного кодування, які регулярно оновлюються для врахування нових технологій та загроз [15].

– Аналіз та тестування безпеки. Застосовуються як статичні, так і динамічні методи аналізу коду для виявлення та усунення вразливостей на ранніх стадіях розробки.

До основних переваг OSSA належить її здатність забезпечувати відповідність міжнародним стандартам безпеки, зокрема FIPS та ISO. Це сприяє зростанню довіри з боку користувачів і полегшує проходження процедур сертифікації. Крім того, впровадження суворих внутрішніх стандартів безпечного кодування та систематичне проведення тестування дозволяє значно зменшити кількість вразливостей у програмних продуктах, що позитивно впливає на їхню надійність і стійкість до кібератак [16].

Водночас OSSA передбачає необхідність постійного оновлення знань та кваліфікації розробників, що потребує додаткових часових і фінансових ресурсів. Це може стати викликом для компаній із обмеженими можливостями щодо безперервного професійного розвитку персоналу.

Методологія рекомендується до застосування в організаціях, що мають на меті інтегрувати заходи безпеки на всіх етапах життєвого циклу розробки ПЗ та прагнуть забезпечити відповідність сучасним стандартам кібербезпеки. Оптимально підходить для великих корпорацій та установ, що використовують рішення Oracle. Типовим прикладом є використання OSSA компанією Oracle у процесах розробки продуктів, таких як Oracle Database та Oracle Cloud Services, що демонструє ефективність цього підходу в контексті забезпечення високого рівня захисту.

OSSA виступає як комплексна методологія, яка дозволяє системно інтегрувати заходи безпеки у процес розробки програмного забезпечення, забезпечуючи відповідність сучасним викликам кібербезпеки та міжнародним нормативним вимогам.

Adobe Secure Product Lifecycle (ASPL) – це комплексна методологія, спрямована на інтеграцію заходів безпеки на всіх етапах життєвого циклу розробки програмного забезпечення. Вона орієнтована на модульний підхід, а саме організацію безпеки при розробці настільних, мобільних та хмарних застосунків.

Основні характеристики ASPL [17]:

– Інтеграція безпеки на всіх етапах розробки. ASPL включає кілька сотень конкретних заходів безпеки, що охоплюють практики розробки ПЗ, процеси та інструменти, забезпечуючи вбудовану безпеку на кожному етапі життєвого циклу продукту.

– Підхід "secure-by-design". Методологія передбачає визначення вимог безпеки на ранніх стадіях розробки, що дозволяє вбудовувати необхідні заходи безпеки ще на етапі проєктування.

– Використання сучасних інструментів тестування. Adobe застосовує автоматизовані засоби для перевірки відповідності застосунків визначеним політикам та контролям безпеки, що забезпечує безпеку протягом усього циклу розробки ПЗ.

ASPLC складається з восьми ключових етапів (рис. 6), кожен з яких відіграє важливу роль у забезпеченні безпеки продуктів Adobe [18]:

– Навчання та сертифікація. Цей етап фокусується на підготовці та сертифікації внутрішніх команд розробників щодо процесів ASPL. Він забезпечує обізнаність команд про сучасні загрози та підходи до безпеки ПЗ. Також включає навчання для неінженерних ролей з метою підвищення загальної культури безпеки в компанії.

– Визначення вимог та планування. На цьому етапі проводиться загальна оцінка стану продукту або сервісу та аналіз ризиків. Це дозволяє внести необхідні корективи, враховуючи

поточний ландшафт загроз, і визначити вимоги до безпеки ще до початку розробки.

– Проєктування. Етап передбачає вбудовування механізмів захисту від потенційних загроз безпосередньо в дизайн нових продуктів та сервісів, а також нових функцій у вже наявних продуктах. Це дає змогу покращити безпековий профіль на ранніх стадіях розробки.

– Розробка та тестування. Під час цього етапу застосовуються практики безпечного кодування для уникнення потенційних вразливостей. Код піддається ретельному внутрішньому та сторонньому тестуванню з використанням стандартних фреймворків та автоматизованих інструментів сканування.

– Підготовка та стабілізація. Цей етап гарантує, що продукт або сервіс готовий до випуску. Він передбачає перевірку стійкості коду, масштабованості та стійкості до атак у середовищі, що імітує реальне використання.

– Розгортання. На цьому етапі дотримуються суворих процесів обробки коду та обмеженого доступу, щоб мінімізувати ризик неправильного розгортання та забезпечити безпечний перехід продукту в експлуатацію.

– Операції та моніторинг. Етап включає постійний моніторинг та логування трафіку для забезпечення максимальної доступності серверів та контролю їхнього стану, що дозволяє оперативно реагувати на потенційні інциденти.

– Реагування на зловживання, шахрайство та інциденти. Етап забезпечує готовність команд швидко реагувати на інциденти, співпрацюючи з експертами з безпеки в Adobe Security Coordination Center для ефективного вирішення та мінімізації наслідків.

Рис. 6. Схема етапів ASPLC

ASPLC демонструє низку ключових переваг, які роблять її ефективним інструментом забезпечення безпеки програмного забезпечення [19]. По-перше, вона сприяє дотриманню міжнародних стандартів безпеки, зокрема OWASP Top 10 та CWE/SANS Top 25. Це забезпечує не лише підвищення рівня довіри з боку клієнтів, а й спрощує процес сертифікації продуктів Adobe відповідно до вимог галузі .

По-друге, ASPLC спрямована на суттєве зниження ризиків, пов'язаних із безпекою. Завдяки системному впровадженню суворих стандартів, процедур перевірки коду та регулярного тестування, методологія дозволяє істотно зменшити кількість вразливостей, що підвищує загальну надійність і стійкість продуктів до кіберзагроз.

Водночас, впровадження ASPL потребує певних витрат. Одним із головних викликів є необхідність постійного оновлення знань у сфері безпеки. Розробники повинні регулярно проходити навчання, щоб залишатися в курсі новітніх підходів та стандартів. Це може вимагати додаткових ресурсів, як у вигляді часу, так і фінансових витрат, що варто враховувати при інтеграції ASPL у корпоративне середовище.

Компанія Adobe успішно впроваджує ASPL у розробці своїх продуктів, таких як Adobe Experience Platform, забезпечуючи високий рівень безпеки та відповідність міжнародним стандартам [20].

Загалом, Adobe Secure Product Lifecycle є комплексною методологією, що сприяє створенню надійного та безпечного програмного забезпечення, відповідаючи сучасним вимогам кібербезпеки та стандартам галузі.

В таблиці 1 представлено зведені результати порівняльного аналізу шести провідних підходів до безпечної розробки програмного забезпечення на основі ключових критеріїв впровадження безпеки на різних етапах життєвого циклу розробки ПЗ. Аналіз базується на дев'яти критеріях, які охоплюють ключові аспекти забезпечення безпеки впродовж усього

життєвого циклу розробки програмного забезпечення:

1. Інтеграція безпеки на ранніх етапах. Усі методології підкреслюють важливість раннього впровадження безпеки. MSDL, OSSA та ASPL орієнтовані на забезпечення безпеки ще на стадії проєктування, включаючи моделювання загроз, у той час як SAMM адаптує глибину інтеграції залежно від рівня зрілості організації.

2. Аналіз архітектури та дизайну. Більшість методологій (MSDL, SSDF, CSDL, OSSA, ASPL) передбачають регулярний або початковий аналіз архітектури на наявність вразливостей. Особливо виділяється CSDL, що впроваджує постійний моніторинг на архітектурному рівні з метою мінімізації ризиків.

3. Безпечне кодування. Усі методології демонструють наявність чітко визначених політик безпечного кодування, серед яких переважає обов'язковість використання статичного аналізу (MSDL) та контроль інструментами (CSDL, ASPL). SSDF і SAMM акцентують на підтримці безпечного коду через моніторинг змін або дотримання стандартів.

4. Тестування безпеки. Усі моделі включають тестування безпеки, при цьому OSSA та CSDL акцентують на комбінованому використанні автоматизованого та ручного тестування. SAMM зобов'язує до проведення тестів перед релізом, що свідчить про акцент на валідацію безпеки перед впровадженням.

5. Усі методології підтримують принцип безперервного управління вразливостями. Зокрема, CSDL та OSSA передбачають контроль у реальному часі, а SAMM та SSDF інтегрують коригувальні заходи у післярелізний період.

Таблиця 1

Порівняння методологій з застосуванням принципів SSDLC

Критерії оцінювання	MSDL (Microsoft)	SSDF (NIST)	SAMM (OWASP)	CSDL (Cisco)	OSSA (Oracle)	ASPL (Adobe)
Інтеграція безпеки на ранніх етапах	Навчання і моделювання загроз з самого початку	Включає вимоги безпеки у дизайн на ранніх етапах	Залежить від рівня зрілості	Безпека впроваджується на всіх етапах розробки	Забезпечує інтеграцію безпеки з самого початку розробки	Включає оцінку ризиків і захищене проєктування
Аналіз архітектури та дизайну на вразливості	Передбачено моделювання загроз і аналіз архітектури	Аналіз ризиків на етапі дизайну	Включено у проєктування для аналізу архітектури	Постійний аналіз на архітектурному рівні для мінімізації ризиків	Проєктування безпеки через регулярний аналіз архітектури	Включає розробку протоколів безпеки та аналіз загроз у дизайні
Безпечне кодування	Обов'язковий статичний аналіз і контроль за безпечним кодуванням	Підтримка безпечного кодування, моніторинг змін у коді	Домен реалізації включає дотримання стандартів безпеки кодування	Використання практик безпечного кодування та контроль інструментами	Обов'язкові практики безпечного кодування для мінімізації вразливостей	Використання стандартів і аудитів коду на кожному етапі
Тестування безпеки	Статичний, динамічний аналіз, пенетраційні тести	Рекомендується автоматизоване тестування на всіх етапах	Верифікація включає обов'язкове тестування перед релізом	Постійне тестування безпеки з акцентом на реальні загрози	Автоматизоване та ручне тестування з фокусом на виявлення вразливостей	Включає інтеграцію безпеки в тестування на кожному етапі розробки
Управління вразливостями після впровадження	Включає моніторинг інцидентів і реагування на них	Постійний моніторинг і корекція вразливостей після релізу	Операції включають управління вразливостями після впровадження	Включає контроль за вразливостями у реальному часі	Постійне оновлення і виправлення вразливостей для підтримки безпеки	Регулярне оновлення та реагування на нові загрози
Можливість адаптації до змін у загрозах	Підтримує регулярне оновлення та вдосконалення	Оновлення з урахуванням нових ризиків	Можливе поступове вдосконалення залежно від потреб	Підтримує адаптацію до нових загроз завдяки гнучкій структурі	Постійне оновлення з урахуванням нових загроз	Регулярне адаптування до нових загроз і ризиків

Продовження Таблиці 1

Залежність від рівня зрілості організації	Придатний для структурованих організацій з дисципліною	Гнучка структура, що підходить різним рівням зрілості	Підтримує різні рівні зрілості з можливістю покращення.	Рекомендований для зрілих команд, які можуть слідувати процесам	Підходить організаціям зі зрілими процесами безпеки	Підтримує різні рівні зрілості з поступовою інтеграцією безпеки
Навчання персоналу з питань безпеки	Обов'язкове навчання з безпеки перед початком	Рекомендується навчання персоналу	Управління включає навчання, залежно від зрілості	Постійне підвищення кваліфікації персоналу з питань безпеки	Вимагає постійного навчання та підвищення кваліфікації	Регулярне навчання з безпеки для підвищення компетентності команди
Гнучкість у використанні методології	Чітко визначені вимоги	Адаптується до різних типів проєктів	Можливе поетапне впровадження	Гнучка модель, адаптована для великих проєктів	Процес стандартизований, але деякі етапи адаптуються	Забезпечує стандартизований підхід

6. Адаптація до нових загроз. Методології демонструють високий рівень адаптивності до нових ризиків. MSDL, OSSA та ASPL підтримують регулярне оновлення політик безпеки, тоді як SAMM і SSDF дозволяють гнучке вдосконалення відповідно до еволюції загроз.

7. Залежність від рівня зрілості організації. SAMM, SSDF і ASPL виокремлюються своєю здатністю масштабуватися відповідно до зрілості організацій. У той час як OSSA та CSDL рекомендуються переважно для вже зрілих безпекових процесів.

8. Навчання персоналу з безпеки визнається критично важливим у всіх методологіях. MSDL і OSSA встановлюють обов'язковість такого навчання, тоді як SAMM і SSDF дозволяють варіативність залежно від зрілості організації.

9. Гнучкість методології найбільш виражена в SAMM, SSDF і CSDL, які дозволяють адаптувати підхід до специфіки проєктів. Натомість MSDL демонструє чітко регламентовану структуру, яка підходить здебільшого для дисциплінованих процесів.

В ході дослідження було також розглянуто методології BOSH Security Integration [21] та Cigital's TouchPoints [22], проте обидві з них припинили існування на момент написання статті. В 2024 році BOSH заявили про закриття відділення, яке займається IoT пристроями, для яких була розроблена методологія, а Cigitals перейшла в склад Synopsys в 2016 році.

Висновки

У сучасному світі, де кіберзагрози стають дедалі більш складними та різноманітними, інтеграція безпеки в процес розробки програмного забезпечення є не лише бажаною, а й життєво необхідною. Життєвий цикл безпечної розробки програмного забезпечення надає системний підхід до забезпечення безпеки на всіх етапах розробки, що дозволяє організаціям виявляти і усувати вразливості на ранніх стадіях, знижуючи ризики для кінцевих користувачів і бізнесу в цілому.

Розглянуті методології, такі як MSDL, SSDF, SAMM, CSDL, OSSA та ASPL демонструють різноманітність підходів до реалізації принципів SSDLC. Кожна з цих методологій має свої особливості, переваги та недоліки, що дозволяє організаціям обирати оптимальний шлях впровадження практик безпеки відповідно до своїх потреб та ресурсів.

Адаптація SSDLC та його варіацій не тільки сприяє покращенню безпеки програмного забезпечення, але й підвищує довіру клієнтів та партнерів, що є важливим фактором успішності у конкурентному середовищі. Інтегруючи безпеку в культуру розробки, організації можуть значно знизити ймовірність кіберінцидентів, а також забезпечити стабільний розвиток і відповідність сучасним стандартам безпеки. Тому впровадження SSDLC є ключовим кроком на шляху до успішної і безпечної розробки програмного забезпечення.

СПИСОК ЛІТЕРАТУРИ

1. Liran Tal. Secure Software Development Lifecycle (SSDLC). Snyk Limited. URL: https://snyk.io/learn/secure-sdlc.

2. Полотай, О. І.; Савуляк, Д.; Лагун, А. Е. Створення безпечного життєвого циклу розроблення програмного забезпечення. *Збірник тез доповідей IVМіжнародної науково-практичної конференції ІБІТ 2022 «Інформаційна безпека та інформаційні технології»* м. Львів 30 листопада 2022 року. Львів : НУЛП, 2022. С. 55 – 57.

3. Розробка безпечних контейнерних застосунків з мікросервісною архітектурою / С. Спасітєлєва та ін. *Кібербезпека: освіта, наука, техніка*, м. Київ 28 вересня 2023 року – Київ : КСУ Грінченка, 2023. С. 193–210.

4. What Is the SSDLC (Secure Software Development Life Cycle)? Hacker One. URL: https://www.hackerone.com/knowledge-center/what-ssdlc-secure-software-development-life-cycle.

5. Robert Mazzoli, Daniel Gluck. Microsoft Security Development Lifecycle (SDL). Microsoft. URL: https://learn.microsoft.com/en-us/compliance/assurance/assurance-microsoft-security-development-lifecycle.

6. Recommendations for securing a development lifecycle / Manuela Pichler et al. Microsoft. URL: https://learn.microsoft.com/en-us/power-platform/well-architected/security/secure-development-lifecycle.

7. Secure Software Development Framework. National Institute of Standarts and Technology. URL: https://csrc.nist.gov/projects/ssdf.

8. Montauban Nicolas. What is NIST SSDF and how should you implement it? *Codific*. URL: https://codific.com/what-is-nist-ssdf-and-how-should-you-implement-it.

9. Software Assurance Maturity Model. OWASP. URL: https://owasp.org/www-project-samm/.

10. Montauban Nicolas. OWASP SAMM: A Comprehensive Introduction. *Codific*. URL: https://codific.com/owasp-samm-comprehensive-introduction.

11. Contact Center Enterprise Solution Security White Paper. Cisco. URL: https://www.cisco.com/c/en/us/products/collateral/customer-collaboration/unified-contact-center-enterprise/white-paper-c11-740413.html.

12. Cisco Secure Development Lifecycle (CSDL) / Simone Arena et al. *Cisco Press*, 24 вересня 2024 року. URL: https://www.ciscopress.com/articles/article.asp?p=3145772&seqNum=8.

13. Importance of Software Security Assurance. Oracle. URL: https://www.oracle.com/corporate/security-practices/assurance.

14. Cloud Native Core Security Guide. Oracle. URL: https://docs.oracle.com/en/industries/communications/cloud-native-core/2.2.0/cnc_security/secure-development-practices.html.

15. Coding Standards. Oracle. URL: https://www.oracle.com/corporate/security-practices/assurance/development.

16. Getting Started Guide for Administrators : Oracle Software Security Assurance (OSSA). Oracle. URL: https://docs.oracle.com/en/cloud/saas/enterprise-performance-management-common/cgsad/3_info_security_epm_cloud_ossa.html.

17. Adobe product security. Adobe. URL: https://www.adobe.com/trust/security/product-security.html.

18. Adobe Secure Engineering White Paper. Adobe. URL: https://www.adobe.com/content/dam/cc/us/en/products/adobe-connect/security-page/pdfs/adobe-secure-engineering-wp.pdf.

19. The Adobe Secure Product Lifecycle (SPLC). Adobe. URL: https://www.adobe.com/trust/security/adobe-splc.html.

20. SecurityinAdobeExperiencePlatform. Adobe. URL: https://business.adobe.com/products/experience-platform/security.html.

21. Bosch IoT Device Management – will be discontinued by mid 2024. Bosch. URL: https://docs.bosch-iot-suite.com/device-management/Feature-list.html.

22. Cigital's Whitepapers. Snyk Limited. URL: https://web.archive.org/web/20150415201431/http://www.cigital.com/resources/white-papers/.

Стаття надійшла до редакції 13.06.2025.
Стаття пройшла рецензування 16.06.2025.

Куперштейн Леонід Михайлович – канд. техн. наук, доцент кафедри захисту інформації.

Тарасюк Микола Борисович – студент групи 1БС-24М кафедри захисту інформації факультету інформаційних технологій та комп'ютерної інженерії.

Маліновський Вадим Ігоревич – канд. техн. наук, доцент кафедри захисту інформації.
Вінницький національний технічний університет.