1. **ОГЛЯД СТАНДАРТІВ І ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**
   1. **Життєвий цикл програмного продукту**

На початку 1970-х років спеціалісти вперше заговорили про наступаючу “кризу ПЗ”. Вона була пов'язана з усе більш підсилюючимся впливом ПО на життя людей, внаслідок чого процеси розробки вимагали постійного вдосконалення. Тому була запропонована концепція життєвого циклу розробки ПО (Software life cycle, SLC). Ця модель реалізує послідовність подій, що відбуваються при розробці ПЗ. Визначення циклу SLC так само як і суперечки щодо сенсу його існування, було предметом багатьох розмов і публікацій, пов'язаних з індустрією ПО. Незважаючи на існування різних точок зору, необхідність в документально підтвердженому процесі розробки ПО залишилася настільки ж актуальною . У 1970 році У.У.Ройс (W.W.Royce) провів ідентифікацію декількох стадій в типовому циклі SLC. Саме Ройс і Баррі Боем (Barry Boehm) припустили. що здійснення контролю кожної стадії процесу розробки призведе до поліпшення якості ПО, а також до збільшення продуктивності при розробці програм. Наприклад, робота з проектування інтерфейсів програмного модуля може бути відкладена до тих нір. поки не будуть визначені кінцеві вимоги. Завдяки цьому скорочується кількість можливих переробок. В цьому випадку мова йдеться про каскадну модель циклу SLC (див. Рис. 1.1). Дії по розробці ПО "протікають" на графіку послідовно, від блоку до блоку.



Рисунок 1.1. Каскадна модель розробки ПЗ

Насправді, більшість дій в ході виконання проектів не відбуваються за лінійним законом. Найчастіше розробникам буває необхідно повернутися до попередньої стадії, щоб виконати завдання, які не були в свій час дозволені відповідним чином. Якщо на стадії розробки виявляється відсутність або неправильне формулювання вимог, розробник не просувається вперед, а повертається назад - до стадії специфікацій вимог. Після завершення і корекції специфікацій вимог повторно вводиться і починає виконуватися стадія розробки. Щоб відобразити повторюваний характер розробки ПО, в графіку рис. 1.1 було додано зворотні стрілки, в результаті чого вийшов графік стандартного життєвого індустріального циклу, показаний на рис. 1.2.



Рисунок 1.2. Повторювана каскадна модель життєвого циклу розробки ПЗ (SLC)

**Короткий опис фаз каскадної моделі**

Наведена нижче характеристика являє собою короткий опис кожної фази каскадної моделі (включаючи фази інтеграції):

* дослідження концепції - відбувається дослідження вимог на системному рівні з метою визначення можливості реалізації концепції;
* Процес системного розподілу - може бути пропущений для систем з розробки виключно ПО. Для систем, в яких необхідна розробка як апаратного, так і програмного забезпечення, необхідні функції застосовуються до ПЗ і обладнання відповідно до загальної архітектури системи;
* процес визначення вимог - визначаються програмні вимозі для інформаційної предметної області системи, призначення, лінії поведінки, продуктивність і інтерфейси. (В разі необхідності в процес також включено функціональний розподіл системних вимог до апаратному) 'і програмному)' забезпечення.);
* процес розробки проекту - розробляється і формулнруетсі логічно Послідовна технічна характеристика програмної системи, включаючи структури даних, архітектуру ПО, інтерфейсні уявлення і процесуальну (алгоритмічну) деталізацію;
* процес реалізації - в результаті його виконання ескізний опис ПО перетворюється в повноцінний програмний продукт. При цьому створюється вихідний код, база даних і документація, які лежать в основі фізичного перетворення проекту. Якщо програмний продукт являє собою придбаний пакет прикладних програм, основними діями по його реалізації будуть установка і тестування пакету програм. Якщо програмний продукт розробляється на замовлення, основними діями є програмування і код-тестування;
* процес установки-включає установку ПО, його перевірку і офіційну приймання замовником для операційного середовища;
* процес експлуатації та підтримки - має на увазі запуск користувачем системи і поточне забезпечення, включаючи надання технічної допомоги, обговорення питань, що виникли з користувачем, реєстрацію запитів користувача на модернізацію і внесення змін, а також коригування або усунення помилок;
* процес супроводу - пов'язаний з дозволом програмних помилок, несправностей. збоїв, модернізацією та внесенням змін, що генеруються процесом підтримки. Складається з ітерацій розробки і передбачає зворотний зв'язок по про доставлених інформації про аномалії;
* процес виведення з експлуатації - вивід існуючої системи з її активного використання або шляхом припинення її роботи, або завдяки її заміні ноной системою або модернізованою версією існуючої системи;
* інтегральні завдання - включають початок роботи над проектом, моніторинг проекту та його управління, управління якістю, верифікацію і атестацію, управління конфігурацією, розробку документації і професійну підготовку протягом усього життєвого циклу.

**V-подібна модель життєвого циклу розробки ПЗ**

V-подібна модель була створена з метою допомогти працюючій над проектом команді в плануванні із забезпеченням подальшої можливості тестування системи. У цій моделі особливе значення надається діям, спрямованим на верифікацію і атестацію продукту. Вона демонструє, що тестування продукту обговорюється, проектується і планується на ранніх етапах життєвого циклу розробки. План випробування приймання замовником розробляється на етапі планування, а компоновочного випробування системи -на фазах аналізу, розробки проекту і т.д. Цей процес розробки планів випробування позначений пунктирною лінією між прямокутниками V-подібної моделі.

V-подібна модель, показана на рис. 4.10, була розроблена як різновид каскадної моделі, а значить, успадкувала від неї таку ж послідовну структуру. Кожна наступна фаза починається після завершення отримання результативних даних попередньої фази. Модель демонструє комплексний підхід до визначення фаз процесу розробки ПО. У ній підкреслені взаємозв'язки, що існують між аналітичними фазами і фазами проектування, які передують кодування, після якого йдуть фази тестування. Пунктирні лінії означаают, що ці фази необхідно розглядати паралельно.

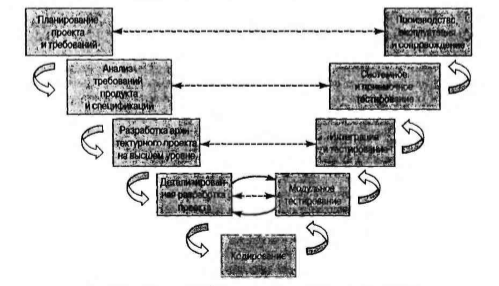


Рисунок 1.X. V-подібна модель життєвого циклу розробки програмного забезпечення

**Фази V-подібної моделі**

Нижче подано короткий опис кожної фази V-подібної моделі, починаючи від планування проекту та вимог аж до приймальних випробувань:

* планування проекту та вимог - визначаються системні вимоги, а також те, яким чином будуть розподілені ресурси організації з метою їх відповідності поставленим вимогам. (В разі необхідності на цій фалі виконується визначення функцій для апаратного і програмного забезпечення);
* аналіз вимог до продукту і його специфікації - аналіз існуючої на даний момент проблеми з ПО, завершується повною специфікацією очікуваної зовнішньої лінії поведінки створюваної програмної системи;
* архітектура або проектування на вищому рівні - визначає, яким чином функції ПО повинні застосуватися при реалізації проекту;
* деталізована розробка проекту - визначає і документально обґрунтовує алгоритми для кожного компонента, який був визначений на фазі побудови архітектури. Ці алгоритми надалі будуть перетворені в код;
* розробка програмного коду - виконується перетворення алгоритмів, визначених на етапі деталізований проектування, в готове ПО;
* модульне тестування - виконується перевірка кожного закодуваного модуля на наявність помилок;
* інтеграція і тестування - установка взаємозв'язків між групами раніше поелементно випробуваних модулів з метою підтвердження того, що ці групи працюють так само добре, як і модулі, випробувані незалежно один від одного на етапі поелементного тестування;
* системне і приймальне тестування - виконується перевірка функціонування програмної системи в цілому (повністю інтегрована система), після приміщенні в її апаратне середовище відповідно до спеціфікацией вимог до ПО.
* виробництво, експлуатація та супровід - ПО запускається у виробництво. На цій фазі передбачені також модернізації та внесення поправок;
* приймальні випробування (на рис. Не показані) - дозволяє користувачеві протестувати функціональні можливості системи на відповідність вихідним вимогам. Після остаточного тестування ПО і довколишній апаратне забезпечення стають робочими. Після цього забезпечується супровід системи.

**Функції забезпечення якості зазвичай включають такі види діяльності:**

* Планування, що полягає в підготовці плану робіт, в якому знаходять відображення вимоги до якості, розподіляються завдання між виконавцями, складаються графіки і розділяється відповідальність.
* Розробку стратегії, методів і процедур, що складається в створенні нормативних інструкцій, що охоплюють всі етапи проектування, в тому числі, встановлення вимог до системи, програмування, тестування відповідно до потреб конкретного проекту. **Ключовим моментом цієї діяльності є можливо більш раннє увагу до питань якості.**
* Розвиненіша коштів, призначених для досягнення високої якості програмного забезпечення, які передбачають адаптацію наявних і створення нових ручних і автоматичних процедур, що дозволяють контролювати його відповідність встановленим вимогам, що стосуються ефективності функціонування і стандартів якості.
* Проведення ревізії, що складаються в аналізі процедур проектування та нормативної документації з точки зору їх відповідності встановленим планам стандартизації процесу розробки програмного забезпечення, в стеженні за виконанням цих планів і наявністю коректує документації.
* Спостереження за випробуваннями програмних засобів, що припускає складання звітів про випробування з аналізом проблем, причин помилок і обгрунтованому коригувальних дій.
* Збереження робочої документації, що стосується звітів про проблеми проектування і програмування системи, контрольних прикладів, протоколів випробувань, тестових даних, оглядів якості та інших аспектів.
  1. **ОГЛЯД** **СТАНДАРТІВ**

~~Якість програмого забезпечення це абстрактне поняття. Її присутність не завжди легко помітити, але відсутність помічається відразу.~~

~~Згідно до міжнародного стандарту ISO 9126[1] якість програмного додатку можна описати, як спроможність програмного продукту при заданих умовах відповідати встановленим та очікуваним вимогам зацікавлених сторін.~~

При розгляді якості програмного забезпечення з точки зору того ж стандарту ISO 9126 [1-4] можна виділити поняття внутрішньої якості, яка пов’язана з характеристиками ПЗ самого по собі, без урахування його поведінки, зовнішньої якості, що характеризує ПЗ з точки зору його поведінки, та якості ПЗ при використанні в різних умовах – та якість, яку відчуває користувач за конкретних сценаріїв роботи ПЗ. Для всіх цих поглядів на якість було введено метрики, які дозволяють оцінити його. Також при розробці якісного ПЗ важливою є якість технологічних процесів його розробки. Відношення між цими складовими якості по схемі, прийнятій в ISO 9126, показано на Рис. 1.1.



Рисунок 1.1- Основні аспекти якості згідно з ISO 9126.

Стандарт ISO 9126 використовує для опису внутрішньої та зовнішньої якості ПЗ багаторівневу модель. На верхньому рівні виділено 6 основних характеристик якості ПЗ. Кожна характеристика описана за допомогою атрибутів, що до неї входять. У кожного атрибута є набір метрик, що дозволяє оцінити цей атрибут. На Рис. 1.2 наведено набір характеристик та атрибутів , визначених в ISO 9126.



Рисунок 1.2. Характеристики та атрибути якості ПЗ відповідно до ISO 9126

Нижче приведено визначення цих характеристик та атрибутів по стандарту ISO 9126:2001.

* Функціональність (functionality). Здатність ПЗ в певних умовах вирішувати завдання, потрібні користувачам. Визначає, що саме робить ПЗ, які завдання воно вирішує
  + - Функціональна придатність (suitability). Здатність вирішувати потрібний набір завдань.
    - Точність (accuracy). Здатність видавати потрібні результати.\
    - Здатність до взаємодії (interoperability). Здатність взаємодіяти з потрібним набором інших систем.
    - Відповідність стандартам і правилам (compliance). Відповідність ПО наявними індустріальним стандартам, нормативним і законодавчим актам, іншим регулюючим нормам.
    - Захищеність (security). Здатність запобігати неавторизований, тобто без зазначення особи, що намагається його здійснити, і не дозволений доступ до даних і програм.
* Надійність (reliability). Здатність ПЗ підтримувати певну працездатність в заданих умовах.
  + Зрілість, завершеність (maturity). Величина, обернена до частоти відмов ПЗ.
  + Стійкість до відмов (fault tolerance) Здатність підтримувати заданий рівень працездатності при відмовах і порушеннях правил взаємодії з оточенням.
  + Здатність до відновлення (recoverability). Здатність відновлювати певний рівень працездатності і цілісність даних після відмови, необхідні для цього час і ресурси.
  + Відповідність стандартам надійності (reliability compliance). Цей атрибут доданий в 2001 році.
* Зручність використання (usability) або практичність. Здатність ПЗ бути зручним в навчанні і використанні, а також привабливим для користувачів.
  + Зрозумілість (understandability). Показник, зворотний до зусиль, витрачених користувачами, щоб сприйняти набір понять, на яких засновано ПО, і їх застосовність для вирішення своїх завдань.
  + Зручність навчання (learnability). Показник, зворотний до зусиль, витрачених користувачами щоб навчитися роботі з ПЗ.
  + Зручність роботи (operability). Показник, зворотний до зусиль, що вживаються користувачами, щоб вирішувати свої завдання за допомогою програмного забезпечення.
  + Відповідність стандартам зручності використання (usability compliance).
* Продуктивність (efficiency) або ефективність. Здатність ПЗ при заданих умовах забезпечувати необхідну працездатність стосовно виділеним для цього ресурсів. Можна визначити її і як відношення одержуваних за допомогою ПЗ результатів до витрачених на це ресурсів.
  + Тимчасова ефективність (time behaviour). Здатність ПЗ видавати очікувані результати, а також забезпечувати передачу необхідного обсягу даних за відведений час.
  + Ефективність використання ресурсів (resource utilisation). Здатність вирішувати потрібні завдання з використанням певних обсягів ресурсів певних видів. Маються на увазі такі ресурси, як оперативна і довгострокова пам'ять, мережеві з'єднання, пристрої введення і виведення, тощо.
  + Відповідність стандартам продуктивності (efficiency compliance).
* Зручність супроводу (maintainability). Зручність проведення всіх видів діяльності, пов'язаних з супровід програм.
  + аналізований (analyzability) або зручність проведення аналізу. Зручність проведення аналізу помилок, дефектів і недоліків, а також зручність аналізу на предмет необхідних змін і їх можливих ефектів.
  + Зручність внесення змін (changeability). Показник, зворотний до трудовитрат на проведення необхідних змін.
  + Зручність перевірки (testability). Показник, зворотний до трудовитрат на проведення тестування та інших видів перевірки того, що внесені зміни привели до потрібних ефектів.
  + Відповідність стандартам зручності супроводу (maintainability compliance).
* Переносимість (portability). Здатність ПЗ зберігати працездатність при перенесенні з одного оточення в інше, включаючи організаційні, апаратні і програмні аспекти оточення.
  + Адаптируемость (adaptability). Здатність ПЗ пристосовуватися до різних оточенням без проведення для цього дій, крім заздалегідь передбачених.
  + Зручність установки (installability). Здатність ПЗ бути встановленим або розгорнутим в певному оточенні.
  + Здатність до співіснування (coexistence). Здатність ПЗ співіснувати з іншими програмами в загальному оточенні, ділячи з ним ресурси.
  + Зручність заміни (replaceability) іншого ПО даними. Здатність ПЗ використовуватися замість іншого ПО для вирішення тих же самих завдань в заданому оточенні.
  + Відповідність стандартам переносимості (portability compliance).
  1. Вимоги до безпеки ПЗ відповідно до ASVS 3.0

Згідно до стандарту ASVS 3.0 запропонованого організацією OWASP можна виділити три рівні перевірки безпеки програмних додатків

Рівень1 призначений для всіх програм.

Рівень 2 призначений для додатків, які містять конфіденційні дані, які вимагають захисту.

Рівень 3 для найбільш важливих додатків - додатків, які **~~виконують високі операції значення~~**, містять конфіденційні медичні дані, або будь-який додаток, яке вимагає високого рівня довіри.

Кожен рівень ASVS містить перелік вимог до безпеки. Кожна з цих вимог також може бути співставлена з функціями безпеки конкретних можливостей, які повинні бути вбудовані в програмне забезпечення розробниками.

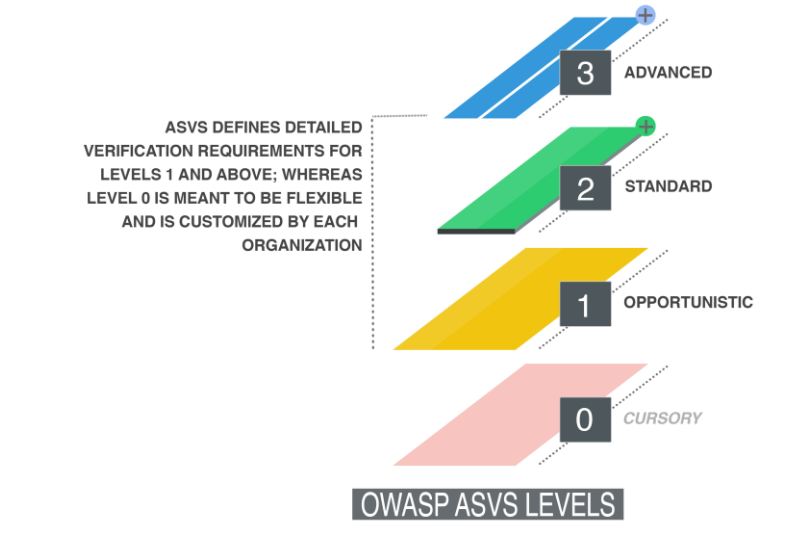


Рисунок x.x Рівні OWASP ASVS 3.0

Додаток досягає ASVS 1-го рівня (або опортуністичних), якщо він адекватно захищає від вразливостей безпеки додатків, які легко виявити, і включені в OWASP Top 10 і інших подібних контрольних списків.

Рівень 1 зазвичай підходить для додатків, де потрібна низька впевненість у правильному використанні засобів контролю безпеки або для швидкого аналізу парку корпоративних додатків або сприяння в розробці пріоритетного списку вимог безпеки в рамках багатофазного зусилля. Елементи управління рівня 1 можуть бути забезпечені або автоматично за допомогою інструментів, або просто вручну без доступу до вихідного коду. Ми розглядаємо рівень 1 як мінімум необхідний для всіх додатків. Загрози для додатка, швидше за все, будуть від зловмисників, які використовують прості і маловитратні методи для виявлення легкодоступних і простих у використанні вразливостей.

Додаток досягає ASVS 2-го рівня (або стандарт), якщо він адекватно захищає від більшості ризиків, пов'язаних з програмним забезпеченням сьогодні. Рівень 2 забезпечує контроль безпеки на місці, ефективний і використовується в додатку. Рівень 2, як правило, підходить для додатків, обробних значні бізнес-бізнес операцій, в тому числі ті, які обробляють інформацію в галузі охорони здоров'я, реалізації бізнес-критичних або чутливих функцій або процесів інших чутливих активів. Загрози застосування 2-го рівня, як правило, будуть кваліфіковані та мотивовані атакуючі зосередивши увагу на конкретних задач з використанням інструментів і методів, які високо практикували і ефективних на виявлення і використання слабких місць в додатках

ASVS Рівень 3 є високий рівень контролю в межах рівня ASVS.This, як правило, зарезервовані для додатків, що вимагають значних рівнів перевірки безпеки, таких, як ті, які можуть бути знайдені в районах військових, охорони здоров'я і безпеки, життєво важливої ​​інфраструктури і т.д. організацій може знадобитися ASVS 3-го рівня для додатків, що виконують критично важливі функції, де збій може істотно вплинути на діяльність організації, і навіть її живучість. Приклад вказівки по застосуванню ASVS Level 3 представлена ​​нижче. Додаток досягає ASVS 3-го рівня (або Advanced), якщо він адекватно захищає від передових вразливостей безпеки додатків, а також демонструє принципи хорошого дизайну безпеки.

Застосування на ASVS 3-го рівня вимагає більш глибокого аналізу, архітектури, кодування і тестування, ніж всі інші рівні. ~~Захищеної applicationis Modularized о~~смислено (для полегшення, наприклад, відмовостійкості, масштабованості і найбільше, рівні безпеки), і кожен модуль (розділених мережевим з'єднанням і / або фізичного примірника) піклується про своїх власних обов'язків безпеки (захист

 в глибину), які повинні бути належним чином задокументовані. ~~Responsibilities~~ включають засоби керування для забезпечення конфіденційності (наприклад, шифрування), цілісність (наприклад, операції, перевірки вхідних даних), доступність

(Наприклад, обробка навантаження коректно), аутентифікацію (в тому числі між системами), безвідмовності, авторизацію і аудит (ведення журналу).

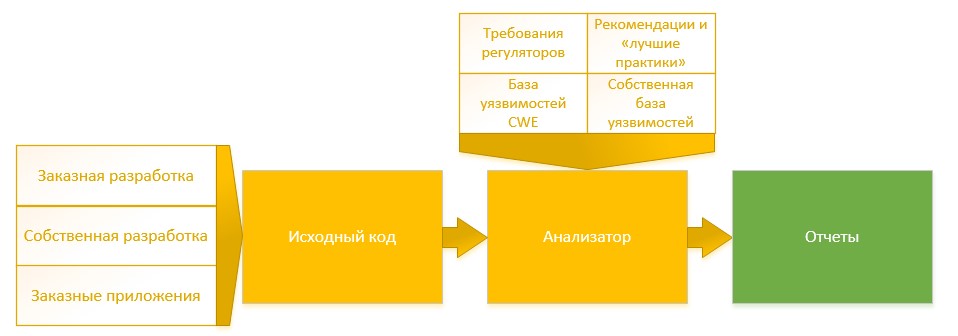
**Класифікація аналізаторів вихідного коду**

Аналізатори вихідного коду - клас програмних продуктів, створених для виявлення і запобігання експлуатації програмних помилок у вихідних кодах. Всі продукти, спрямовані на аналіз вихідного коду, можна умовно розділити на три типи:

-Перша група включає в себе аналізатори коду веб-додатків і засоби щодо запобігання експлуатації вразливостей веб-сайтів.

-Друга група - аналізатори вбудованого коду, що дозволяють виявити проблемні місця в початкових текстах модулів, призначених для розширення функціональності корпоративних і виробничих систем. До таких модулів відносяться програми для лінійки продуктів 1С, розширення CRM-систем, систем управління підприємством і систем SAP.

-Остання група призначена для аналізу вихідного коду на різних мовах програмування, що не відносяться до бізнес-додатків і веб-додатків. Такі аналізатори призначені для замовників і розробників програмного забезпечення. У тому числі дана група аналізаторів застосовується для використання методології захищеної розробки програмних продуктів. Аналізатори статичного коду знаходять проблеми і потенційно вразливі місця в початкових кодах і видають рекомендації для їх усунення. Варто зазначити, що більшість з аналізаторів відносяться до змішаних типів і виконують функції з аналізу широкого спектра програмних продуктів - веб-додатків, вбудованого коду і звичайного програмного забезпечення. Проте в даному огляді упор зроблений на застосування аналізаторів замовниками розробки, тому більша увага приділяється аналізатора веб-додатків і вбудованого коду. Аналізатори можуть містити різні механізми аналізу, але найбільш поширеним і універсальним є статичний аналіз вихідного коду - SAST (Static Application Security Testing), також існують методи динамічного аналізу - DAST (Dynamic Application Security Testing), що виконують перевірки коду при його виконанні, і різні гібридні варіанти, що поєднують різні типи аналізів. Динамічний аналіз є самостійним методом перевірки, який може розширювати можливості статичного аналізу або застосовуватися самостійно в тих випадках, коли доступ до вихідних текстів відсутній. В даному огляді розглядаються тільки статичні аналізатори. Аналізатори вбудованого коду і веб-додатків розрізняються по набору характеристик. У нього входять не тільки якість аналізу і перелік підтримуваних програмних продуктів і мов програмування, а й додаткові механізми: можливість здійснення автоматичного виправлення помилок, наявність функцій щодо запобігання експлуатації помилок без змін коду, можливість поновлення вбудованої бази вразливостей і помилок програмування, наявність сертифікатів відповідності та можливість виконання вимог різних регуляторів. Принципи роботи аналізаторів вихідного коду Загальні принципи роботи схожі для всіх класів аналізаторів: і аналізаторів вихідного коду веб-додатків, і аналізаторів вбудованого коду. Відмінність між цими типами продуктів - тільки в можливості визначити особливості виконання і взаємодії коду із зовнішнім світом, що відбивається в базах вразливостей аналізаторів. Велика частина аналізаторів, представлених на ринку, виконує функції обох класів, однаково добре перевіряючи як вбудовується в бізнес-додатки код, так і код веб-додатків. Вхідними даними для аналізатора вихідного коду є масив вихідних текстів програм і його залежностей (підвантажуваних модулів, використовуваного стороннього програмного забезпечення і т. Д.). Як результатів роботи всі аналізатори видають звіт про виявлені вразливості і помилки програмування, додатково деякі аналізатори надають функції по автоматичному виправленню помилок. Варто зазначити, що автоматичне виправлення помилок не завжди працює коректно, тому даний функціонал призначений тільки для розробників веб-додатків і вбудованих модулів, замовник продукту повинен спиратися тільки на фінальний звіт аналізатора і використовувати отримані дані для прийняття рішення по прийманню та впровадження розробленого коду або відправки його на доопрацювання.



~~Малюнок 1~~. Алгоритм роботи аналізатора вихідних кодів При проведенні оцінки вихідних текстів аналізатори використовують різні бази даних, що містять опис вразливостей і помилок програмування: Власна база вразливостей і помилок програмування - у кожного розробника аналізаторів вихідних кодів є свої відділи аналітики і досліджень, які готують спеціалізовані бази для аналізу вихідних текстів програм. Якість власної бази - один з ключових критеріїв, що впливає на загальну якість роботи продукту. Крім того, власна база повинна бути динамічною і постійно оновлюваною - нові вектори атак і експлуатації вразливостей, а також зміни в мовах програмування і методи розробки вимагають від розробників аналізаторів виконувати постійні оновлення бази для збереження високої якості перевірки. Продукти зі статичної неоновлюваної базою найчастіше програють в порівняльних тестах. Державні бази помилок програмування - існує ряд державних баз вразливостей, складанням і підтримкою яких займаються регулятори різних країн. Наприклад, в США використовується база CWE - Common Weakness Enumeration, обслуговуванням якої займається організація MITRE, підтримувана в тому числі Міністерством оборони США. Вимоги стандартів і рекомендації по захищеному програмування - існує як ряд державних і галузевих стандартів, що описують вимоги до безпечної розробці свпріложеній, так і ряд рекомендацій і «кращих практик» від світових експертів в області розробки і захисту програмного забезпечення. Дані документи безпосередньо не описують помилки програмування, на відміну від CWE, але містять перелік методів, які можуть бути перетворені для використання в статичному аналізаторі вихідного коду. Від того, які бази використовуються в аналізаторі, безпосередньо залежить якість проведення аналізу, кількість помилкових спрацьовувань і пропущених помилок. Крім того, аналіз на відповідність вимогам регуляторів дозволяє полегшити і спростити процедуру зовнішнього аудиту інфраструктури та інформаційної системи в тому випадку, якщо вимоги є обов'язковими. Наприклад, вимоги PCI DSS обов'язкові для веб-додатків і вбудованого коду, що працює з платіжною інформацією по банківських картах, при цьому проведення зовнішнього аудиту з виконання PCI DSS здійснюється в тому числі з аналізом застосовуваних програмних продуктів.