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 ескізне 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.

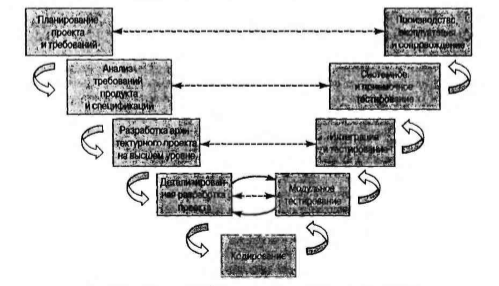


Рисунок 1.X. V-подібна модель життєвого циклу розробки програмного забезпечення

**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).

**Классификация анализаторов исходного кода**

Анализаторы исходного кода — класс программных продуктов, созданных для выявления и предотвращения эксплуатации программных ошибок в исходных кодах. Все продукты, направленные на анализ исходного кода, можно условно разделить на три типа: Первая группа включает в себя анализаторы кода веб-приложений и средства по предотвращению эксплуатации уязвимостей веб-сайтов. Вторая группа — анализаторы встраиваемого кода, позволяющие обнаружить проблемные места в исходных текстах модулей, предназначенных для расширения функциональности корпоративных и производственных систем. К таким модулям относятся программы для линейки продуктов 1С, расширения CRM-систем, систем управления предприятием и систем SAP. Последняя группа предназначена для анализа исходного кода на различных языках программирования, не относящихся к бизнес-приложениям и веб-приложениям. Такие анализаторы предназначены для заказчиков и разработчиков программного обеспечения. В том числе данная группа анализаторов применяется для использования методологии защищенной разработки программных продуктов. Анализаторы статического кода находят проблемы и потенциально уязвимые места в исходных кодах и выдают рекомендации для их устранения. Стоит отметить, что большинство из анализаторов относятся к смешанным типам и выполняют функции по анализу широкого спектра программных продуктов — веб-приложений, встраиваемого кода и обычного программного обеспечения. Тем не менее в данном обзоре упор сделан на применение анализаторов заказчиками разработки, поэтому большее внимание уделяется анализаторам веб-приложений и встраиваемого кода. Анализаторы могут содержать различные механизмы анализа, но наиболее распространенным и универсальным является статический анализ исходного кода — SAST (Static Application Security Testing), также существуют методы динамического анализа — DAST (Dynamic Application Security Testing), выполняющие проверки кода при его исполнении, и различные гибридные варианты, совмещающие разные типы анализов. Динамический анализ является самостоятельным методом проверки, который может расширять возможности статического анализа или применяться самостоятельно в тех случаях, когда доступ к исходным текстам отсутствует. В данном обзоре рассматриваются только статические анализаторы. Анализаторы встраиваемого кода и веб-приложений различаются по набору характеристик. В него входят не только качество анализа и перечень поддерживаемых программных продуктов и языков программирования, но и дополнительные механизмы: возможность осуществления автоматического исправления ошибок, наличие функций по предотвращению эксплуатации ошибок без изменений кода, возможность обновления встроенной базы уязвимостей и ошибок программирования, наличие сертификатов соответствия и возможность выполнения требований различных регуляторов.   Принципы работы анализаторов исходного кода Общие принципы работы схожи для всех классов анализаторов: и анализаторов исходного кода веб-приложений, и анализаторов встраиваемого кода. Отличие между этими типами продуктов — только в возможности определить особенности выполнения и взаимодействия кода с внешним миром, что отражается в базах уязвимостей анализаторов. Большая часть анализаторов, представленных на рынке, выполняет функции обоих классов, одинаково хорошо проверяя как встраиваемый в бизнес-приложения код, так и код веб-приложений. Входными данными для анализатора исходного кода является массив исходных текстов программ и его зависимостей (подгружаемых модулей, используемого стороннего программного обеспечения и т. д.). В качестве результатов работы все анализаторы выдают отчет об обнаруженных уязвимостях и ошибках программирования, дополнительно некоторые анализаторы предоставляют функции по автоматическому исправлению ошибок. Стоит отметить, что автоматическое исправление ошибок не всегда работает корректно, поэтому данный функционал предназначен только для разработчиков веб-приложений и встраиваемых модулей, заказчик продукта должен опираться только на финальный отчет анализатора и использовать полученные данные для принятия решения по приемке и внедрению разработанного кода или отправки его на доработку.   Рисунок 1. Алгоритм работы анализатора исходных кодов   При проведении оценки исходных текстов анализаторы используют различные базы данных, содержащие описание уязвимостей и ошибок программирования: Собственная база уязвимостей и ошибок программирования — у каждого разработчика анализаторов исходных кодов есть свои отделы аналитики и исследований, которые готовят специализированные базы для анализа исходных текстов программ. Качество собственной базы — один из ключевых критериев, влияющий на общее качество работы продукта. Кроме того, собственная база должна быть динамической и постоянно обновляемой — новые векторы атак и эксплуатации уязвимостей, а также изменения в языках программирования и методах разработки требуют от разработчиков анализаторов выполнять постоянные обновления базы для сохранения высокого качества проверки. Продукты со статической необновляемой базой чаще всего проигрывают в сравнительных тестах. Государственные базы ошибок программирования — существует ряд государственных баз уязвимостей, составлением и поддержкой которых занимаются регуляторы разных стран. К примеру, в США используется база CWE — Common Weakness Enumeration, обслуживанием которой занимается организация MITRE, поддерживаемая в том числе Министерством обороны США. В России пока отсутствует аналогичная база, но ФСТЭК России в будущем планирует дополнить свои базы уязвимостей и угроз базой по ошибкам программирования. Анализаторы уязвимостей реализуют поддержку базы CWE, встраивая ее в собственную базу уязвимостей или используя как отдельный механизм проверки. Требования стандартов и рекомендации по защищенному программированию — существует как ряд государственных и отраслевых стандартов, описывающих требования к безопасной разработке свприложений, так и ряд рекомендаций и «лучших практик» от мировых экспертов в области разработки и защиты программного обеспечения. Данные документы напрямую не описывают ошибки программирования, в отличие от CWE, но содержат перечень методов, которые могут быть преобразованы для использования в статическом анализаторе исходного кода. От того, какие базы используются в анализаторе, напрямую зависит качество проведения анализа, количество ложных срабатываний и пропущенных ошибок. Кроме того, анализ на соответствие требованиям регуляторов позволяет облегчить и упросить процедуру внешнего аудита инфраструктуры и информационной системы в том случае, если требования являются обязательными. К примеру, требования PCI DSS обязательны для веб-приложений и встраиваемого кода, работающего с платежной информацией по банковским картам, при этом проведение внешнего аудита по выполнению PCI DSS осуществляется в том числе с анализом применяемых программных продуктов.