**Вопрос 1: МДК 04.02**

**Многоуровневая модель качества программного обеспечения: Как она используется для оценки качества? Какие уровни включает эта модель, и как они взаимосвязаны? Объясните с примерами реальных программных систем для внешнего, внутреннего и операционного уровней.**

**1. Концепция качества программного обеспечения.** Качество программного обеспечения (ПО) – это совокупность характеристик и свойств ПО, которые определяют его способность удовлетворять заявленные и подразумеваемые потребности пользователей. Многоуровневая модель качества позволяет структурировать оценку качества и разделить его на отдельные уровни, которые рассматривают разные аспекты продукта.

**Задача модели:** Обеспечить всестороннюю оценку качества ПО, начиная от внутренней структуры и заканчивая поведением при эксплуатации.

**2. Уровни многоуровневой модели качества.** Модель качества разделяется на три основных уровня: **внешний**, **внутренний** и **операционный**. Они позволяют оценивать качество на разных стадиях жизненного цикла программного обеспечения.

1. **Внешний уровень качества**
   * Оценивает поведение программного обеспечения с точки зрения пользователя.
   * **Характеристики:** функциональность, удобство использования, производительность, надёжность.
   * **Пример:**
     + Если пользователь взаимодействует с приложением, он ожидает, что кнопки работают корректно, данные обрабатываются быстро, а интерфейс понятен. Например, в мобильном приложении банковского сервиса пользователь ожидает корректное выполнение перевода и быстроту отклика.
2. **Внутренний уровень качества**
   * Рассматривает структуру программного кода и архитектуру приложения.
   * **Характеристики:** читаемость кода, модульность, поддерживаемость, соответствие стандартам разработки.
   * **Пример:**
     + Если код плохо структурирован, он становится трудным для поддержки. Например, если разработчик использует повторяющиеся фрагменты кода вместо функций или нарушает принцип DRY (Don't Repeat Yourself), это ухудшает внутреннее качество.
3. **Операционный уровень качества**
   * Оценивает поведение программы при реальной эксплуатации в рабочей среде.
   * **Характеристики:** стабильность работы, масштабируемость, безопасность, совместимость.
   * **Пример:**
     + Программное обеспечение может показывать отличные результаты в тестовой среде, но при работе под высокой нагрузкой может «падать». Например, веб-сервер должен корректно обрабатывать большое количество запросов от пользователей во время пиковых нагрузок.

**3. Взаимосвязь уровней качества**

Уровни модели качества тесно связаны между собой:

* Внутреннее качество (чистота кода и архитектура) напрямую влияет на внешнее качество (поведение программы для пользователя).
* Операционное качество является результатом правильного подхода к внутреннему и внешнему качеству на этапах разработки и тестирования.

**Пример взаимосвязи:**

* Хорошая внутренняя архитектура кода способствует уменьшению ошибок на внешнем уровне (корректная работа функций).
* Стабильное внешнее качество влечёт за собой хорошее операционное поведение программы при нагрузках.

**Вывод**

Многоуровневая модель качества позволяет разработчикам и тестировщикам всесторонне оценивать ПО, выявлять проблемы на разных этапах разработки и улучшать его свойства. Эта модель помогает понять, что качество – это не только внешний результат, но и внутренняя структура, которая обеспечивает стабильность на этапе эксплуатации.

Объяснение ошибок:

**Первичная ошибка**: Неправильное условие в цикле.

В строке:for (size\_t i = 0; i <= numbers.size(); i++)

Цикл выполняется до i <= numbers.size(), что вызывает обращение к индексу за пределами массива на последней итерации.

**Последствие:** Ошибка времени выполнения (segmentation fault).

**Вторичная ошибка: Неправильное вычисление среднего значения.**

В строке:

return sum / numbers.size();

Выполняется целочисленное деление, что приводит к потере дробной части результата.

**Последствие:** Программа возвращает неверный результат.

Исправленный код:

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

class FileProcessor {

private:

std::string inputFile;

std::string outputFile;

public:

void setFiles(const std::string& inFile, const std::string& outFile) {

inputFile = inFile;

outputFile = outFile;

}

double calculateAverageFromFile() {

std::ifstream inFile(inputFile);

if (!inFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть файл для чтения.");

}

std::vector<int> numbers;

std::string line;

while (std::getline(inFile, line)) {

std::stringstream ss(line);

int num;

while (ss >> num) {

numbers.push\_back(num);

}

}

if (numbers.empty()) {

throw std::runtime\_error("Файл не содержит чисел для обработки.");

}

int sum = 0;

for (size\_t i = 0; i < numbers.size(); i++) { // Исправлено условие

sum += numbers[i];

}

return static\_cast<double>(sum) / numbers.size(); // Исправлено для корректного деления

}

void writeResultToFile(double result) {

std::ofstream outFile(outputFile);

if (!outFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть файл для записи.");

}

outFile << "Среднее арифметическое: " << result << std::endl;

}

};

int main() {

try {

FileProcessor processor;

processor.setFiles("input.txt", "output.txt");

double average = processor.calculateAverageFromFile();

processor.writeResultToFile(average);

std::cout << "Результат сохранён в файл." << std::endl;

} catch (const std::exception& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

return 0;

}

**Вопрос 2 МДК 04.02**

**Какие аспекты качества программного обеспечения рассматривает многоуровневая модель? Как эта модель может быть использована на практике при разработке программного продукта? Приведите пример её применения на этапе тестирования или внедрения.**

**1. Аспекты качества программного обеспечения**

В многоуровневой модели качества учитываются несколько ключевых аспектов:

1. **Функциональность** – способность ПО выполнять заявленные функции.
2. **Надёжность** – устойчивость к ошибкам и стабильность работы.
3. **Производительность** – время выполнения операций и отклика.
4. **Удобство использования** – пользовательский интерфейс и удобство работы.
5. **Безопасность** – защита данных от угроз и атак.
6. **Поддерживаемость** – лёгкость внесения изменений и расширений.

**2. Применение модели на практике**

Многоуровневая модель качества может применяться на разных этапах жизненного цикла ПО, включая разработку, тестирование и внедрение.

**Сценарий:** Проверка функциональности и надёжности в процессе тестирования.

* **Внешний уровень:**

При тестировании продукта, например, интернет-магазина, проверяются такие параметры, как:

* + Корректность оформления заказов.
  + Быстрота работы страницы корзины.
  + Правильность расчёта стоимости товаров.
* **Внутренний уровень:**

Для обеспечения надёжности на уровне кода важно:

* + Провести анализ кода с использованием инструментов статического анализа.
  + Проверить, что модули имеют минимальное количество взаимозависимостей.
  + Оптимизировать алгоритмы для повышения производительности.
* **Операционный уровень:**

После внедрения проводится нагрузочное тестирование:

* + Например, имитация одновременной работы 1000 пользователей, чтобы проверить, выдерживает ли сервер такую нагрузку.
  + Важно также учесть обработку ошибок: если один модуль отказал, система должна продолжить работу.

**3. Пример применения модели**

**Ситуация:** Разработка корпоративного веб-приложения для управления задачами (Task Manager).

1. **Внешний уровень:**
   * Пользователь должен видеть корректно работающий интерфейс:
     + Кнопки «Добавить задачу», «Удалить» и «Редактировать» работают без задержек.
2. **Внутренний уровень:**
   * Код должен быть чистым и структурированным:
     + Например, каждая функция имеет чёткое предназначение, код легко читается.
     + Используются автоматические тесты для проверки логики работы модулей.
3. **Операционный уровень:**
   * При большой нагрузке (500 активных пользователей) программа должна корректно работать:
     + Сервер не перегружается.
     + База данных обрабатывает все запросы без потери данных.

**Вывод**

Многоуровневая модель качества ПО помогает анализировать и улучшать продукт на всех этапах жизненного цикла, начиная с разработки и заканчивая эксплуатацией. Правильное применение этой модели в тестировании и внедрении позволяет обеспечить стабильную и надёжную работу программного обеспечения, удовлетворяя требования пользователей и бизнеса.

**Объяснение ошибок:**

1. **Первичная ошибка: Неполная проверка формата даты.**

Код проверяет только наличие разделителей '/' между компонентами даты.

**Последствие:** Неверный формат, например, "32/13/2022" или "abc/12/2022", не будет корректно обрабатываться.

1. **Вторичная ошибка: Неправильный расчёт разницы дней.**

Логика вычисления количества дней не учитывает високосные годы.

**Последствие:** Результат будет неверным для дат, включающих високосные годы.

Исправленный код:

#include <iostream>

#include <fstream>

#include <sstream>

#include <stdexcept>

#include <string>

#include <cmath>

class Date {

public:

int day, month, year;

Date(int d, int m, int y) : day(d), month(m), year(y) {

if (!isValid()) {

throw std::runtime\_error("Некорректная дата: " + std::to\_string(d) + "/" + std::to\_string(m) + "/" + std::to\_string(y));

}

}

static Date parse(const std::string& dateStr) {

std::stringstream ss(dateStr);

int d, m, y;

char delim1, delim2;

ss >> d >> delim1 >> m >> delim2 >> y;

if (delim1 != '/' || delim2 != '/' || ss.fail()) {

throw std::runtime\_error("Некорректный формат даты.");

}

return Date(d, m, y);

}

private:

bool isValid() {

if (month < 1 || month > 12 || day < 1) return false;

static const int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

int maxDays = daysInMonth[month - 1];

if (month == 2 && isLeapYear(year)) {

maxDays = 29;

}

return day <= maxDays;

}

static bool isLeapYear(int y) {

return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);

}

};

class DateProcessor {

public:

static int daysBetween(const Date& start, const Date& end) {

int days1 = calculateDays(start.year, start.month, start.day);

int days2 = calculateDays(end.year, end.month, end.day);

return abs(days2 - days1);

}

private:

static int calculateDays(int year, int month, int day) {

static const int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

int totalDays = year \* 365 + day;

for (int i = 0; i < month - 1; ++i) {

totalDays += daysInMonth[i];

}

totalDays += countLeapYears(year, month);

return totalDays;

}

static int countLeapYears(int year, int month) {

if (month <= 2) year--;

return year / 4 - year / 100 + year / 400;

}

};

int main() {

try {

std::ifstream inFile("dates.txt");

if (!inFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть файл.");

}

std::string dateStr1, dateStr2;

std::getline(inFile, dateStr1);

std::getline(inFile, dateStr2);

Date date1 = Date::parse(dateStr1);

Date date2 = Date::parse(dateStr2);

int days = DateProcessor::daysBetween(date1, date2);

std::cout << "Количество дней между датами: " << days << std::endl;

} catch (const std::exception& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

return 0;

}

**Вопрос 3 МДК 04.02**

**Объекты уязвимостей в программном обеспечении: Какие существуют их основные виды и почему они представляют угрозу для безопасности системы? Приведите примеры объектов уязвимостей для каждого типа и объясните их влияние на защиту.**

**Ответ:**

**Объекты уязвимостей** — это компоненты программного обеспечения или информационной системы, которые подвержены атакам из-за ошибок разработки, некорректной конфигурации, устаревших технологий или неосторожности пользователей. Эти уязвимости могут быть использованы злоумышленниками для нарушения целостности, конфиденциальности и доступности данных.

**Основные виды объектов уязвимостей:**

1. **Исходный код**: Ошибки в написании кода приводят к уязвимостям. Проблемы могут быть связаны с некорректной обработкой данных, недостаточной валидацией ввода или неправильным использованием функций.
   * **Пример**: SQL-инъекции, возникающие из-за неэкранированного ввода данных.

Например: SELECT \* FROM users WHERE id = '1 OR 1=1';

1. **Конфигурационные файлы**: Неправильная настройка системы может привести к утечке данных или проникновению в систему.
   * **Пример**: Открытый доступ к файлам конфигурации с хранимыми паролями или ключами API.
2. **Сетевые интерфейсы и протоколы**: Ошибки в сетевых настройках и протоколах могут использоваться для атаки на системы через интернет.
   * **Пример**: Использование незашифрованного соединения HTTP вместо HTTPS позволяет перехватывать трафик злоумышленникам (атака "man-in-the-middle").
3. **Базы данных**: Неоптимальная защита баз данных может привести к утечке информации.
   * **Пример**: Недостаточная аутентификация в SQL-базах позволяет атакующему получить несанкционированный доступ.
4. **Операционные системы и приложения**: Устаревшие версии ПО содержат уязвимости, которые могут быть использованы для взлома.
   * **Пример**: Уязвимость EternalBlue в Windows позволила распространять вирус WannaCry в 2017 году.
5. **Пользователи**: Ошибки пользователей, такие как слабые пароли или игнорирование обновлений, также являются объектами уязвимостей.
   * **Пример**: Использование пароля "123456" или отсутствие двухфакторной аутентификации.

**Почему объекты уязвимостей представляют угрозу?**

* Они могут использоваться злоумышленниками для кражи, модификации или уничтожения данных.
* Уязвимости приводят к нарушению работоспособности системы, финансовым потерям и репутационным рискам.

**Объяснение ошибок:**

**Первичная ошибка:** В конструкторе класса Rectangle ширина (width) и высота (height) допускают отрицательные значения.

**Последствие:** Площадь и периметр вычисляются некорректно, что может вызвать ошибки в логике программы.

**Вторичная ошибка:** Формула расчёта площади содержит лишний множитель 0.5:

return width \* height \* 0.5;

**Последствие:** Площадь всегда рассчитывается неверно.

Исправленный код:

#include <iostream>

#include <stdexcept>

class Rectangle {

private:

double width;

double height;

public:

Rectangle(double w, double h) {

if (w <= 0 || h <= 0) {

throw std::invalid\_argument("Ширина и высота должны быть положительными числами.");

}

width = w;

height = h;

}

double calculateArea() const {

return width \* height; // Исправлено: убран лишний множитель

}

double calculatePerimeter() const {

return 2 \* (width + height);

}

void displayProperties() const {

std::cout << "Ширина: " << width << ", Высота: " << height << std::endl;

std::cout << "Площадь: " << calculateArea() << std::endl;

std::cout << "Периметр: " << calculatePerimeter() << std::endl;

}

};

int main() {

try {

Rectangle rect(5, 10); // Исправлено: положительные значения

rect.displayProperties();

} catch (const std::exception& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

return 0;

}

**Исправления:**

**Проверка входных данных:**

Добавлена проверка в конструктор:

if (w <= 0 || h <= 0) {

throw std::invalid\_argument("Ширина и высота должны быть положительными числами.");

}

Это предотвращает создание объекта с некорректными значениями.

**Корректная формула площади:** Формула изменена на: return width \* height;

**Улучшение обработки ошибок:**

Добавлено исключение с понятным сообщением: throw std::invalid\_argument("Ширина и высота должны быть положительными числами.");

**Пояснение исправлений:**

**Первичная ошибка:** Отрицательные значения ширины и высоты теперь обрабатываются через исключения.

**Вторичная ошибка:** Формула площади исправлена, чтобы результат был корректным.

**Вопрос 4 МДК 04.02**

**Какие основные факторы способствуют появлению уязвимостей в программном обеспечении? Как их можно предотвратить на различных этапах жизненного цикла разработки программного продукта?**

**Ответ:**

**Факторы, приводящие к уязвимостям в программном обеспечении:**

1. **Ошибки на этапе проектирования**:
   * Отсутствие учёта требований безопасности при разработке архитектуры.
   * Пример: отсутствие планирования по валидации пользовательского ввода.
2. **Ошибки в коде**:
   * Неправильная обработка данных, некорректные условия, отсутствие проверки ввода.
   * Пример: переполнение буфера из-за неправильно заданного размера массива.
3. **Неправильная конфигурация системы**:
   * Использование небезопасных настроек по умолчанию.
   * Пример: открытые сетевые порты или слабые пароли администраторов.
4. **Устаревшие версии программного обеспечения**:
   * Отсутствие обновлений и патчей безопасности для устранения известных уязвимостей.
   * Пример: использование старой версии OpenSSL с уязвимостью Heartbleed.
5. **Человеческий фактор**:
   * Пользователи могут стать причиной уязвимостей из-за ошибок или недостатка знаний.
   * Пример: открытие фишинговых писем или скачивание заражённых файлов.

**Методы предотвращения уязвимостей на этапах жизненного цикла разработки ПО:**

1. **Этап проектирования**:
   * Учёт требований безопасности в архитектуре.
   * Проведение анализа рисков.
   * Использование принципов **Secure by Design**.
2. **Этап кодирования**:
   * Применение стандартов кодирования и практик безопасной разработки.
   * Использование статического анализа кода для выявления ошибок (например, SonarQube).
   * Регулярные ревью кода для обнаружения логических и синтаксических ошибок.
3. **Этап тестирования**:
   * Проведение статического и динамического анализа безопасности.
   * Пентесты и тестирование на проникновение с использованием инструментов, таких как Burp Suite или OWASP ZAP.
4. **Этап внедрения**:
   * Регулярное обновление ПО и системных компонентов.
   * Настройка безопасных параметров конфигурации.
   * Защита от несанкционированного доступа (сильные пароли, двухфакторная аутентификация).
5. **Этап эксплуатации**:
   * Мониторинг безопасности и аудит логов.
   * Настройка систем обнаружения и предотвращения вторжений (IDS/IPS).
   * Регулярное обучение персонала по вопросам информационной безопасности.

Уязвимости возникают на всех этапах жизненного цикла ПО и могут быть вызваны как техническими ошибками, так и человеческим фактором. Применение комплексного подхода к обеспечению безопасности на всех стадиях разработки позволяет минимизировать риски и защитить программное обеспечение от атак.

**Вопрос 5 МДК 04.02**

**Дестабилизирующие факторы в программном обеспечении: Какие основные категории дестабилизирующих факторов можно выделить, и как они влияют на надёжность системы? Опишите, как внешние, внутренние и системные факторы могут привести к сбоям, и приведите примеры каждой категории с пояснением их последствий для функционирования программного продукта.**

**Ответ:**

**Дестабилизирующие факторы** — это воздействия или условия, которые ухудшают работу программного обеспечения, приводят к сбоям, снижению производительности или полной недоступности системы. Эти факторы могут возникать на разных этапах разработки и эксплуатации программного обеспечения.

**Классификация дестабилизирующих факторов:**

1. **Технические факторы.** Возникают из-за проблем в программном коде, архитектуре или устаревших компонентах.
   * **Ошибки в коде:** Недочёты при программировании приводят к логическим и функциональным сбоям. *Пример:* В 1996 году ракета **Ariane 5** взорвалась из-за переполнения памяти при преобразовании данных.
   * **Сложность архитектуры:** Чем сложнее архитектура, тем больше точек отказа. *Пример:* Сбой в микросервисной архитектуре одной службы может нарушить работу всей системы.
   * **Устаревшие компоненты:** Использование старых библиотек с известными уязвимостями. *Пример:* Атака **Heartbleed** из-за уязвимости в старой версии OpenSSL.
2. **Человеческие факторы.** Связаны с ошибками, допущенными программистами, администраторами и пользователями.
   * **Ошибки программистов:** Некорректное написание или отсутствие тестирования кода. *Пример:* Некорректное преобразование данных стало причиной сбоя ракеты Ariane 5.
   * **Ошибки администраторов:** Неправильные настройки серверов или маршрутизации. *Пример:* Сбой Facebook в 2021 году произошел из-за ошибки в конфигурировании протокола BGP.
   * **Неправильная эксплуатация:** Пользователи могут вводить некорректные данные или использовать систему не по назначению. *Пример:* В медицинской системе неверный ввод данных привёл к сбою анализа.
3. **Окружение и инфраструктура.** Факторы, связанные с аппаратными и внешними условиями эксплуатации.
   * **Сбои оборудования:** Отказы серверов, дисков или сетевых компонентов. *Пример:* Сбой в дата-центре **AWS** вызвал недоступность множества сервисов.
   * **Перебои с питанием:** Электрические сбои приводят к остановке систем. *Пример:* Электрический сбой в Токио в 2015 году привёл к остановке метро.
   * **Природные факторы:** Катастрофы, такие как наводнения или землетрясения. *Пример:* Ураган в 2012 году затопил серверы Google.
4. **Внешние угрозы.** Включают кибератаки и неконтролируемые нагрузки на систему.
   * **Кибератаки:** DDoS-атаки, вирусы и взлом систем. *Пример:* Атака **WannaCry** в 2017 году парализовала сотни тысяч компьютеров.
   * **Неконтролируемая нагрузка:** Резкий рост пользователей может перегрузить систему. *Пример:* В «Чёрную пятницу» сайты интернет-магазинов нередко выходят из строя.

Дестабилизирующие факторы — это многогранные угрозы для надёжности системы. Их идентификация и устранение на всех этапах разработки и эксплуатации позволяет минимизировать риски и повышать устойчивость программного обеспечения.

**Вопрос 6 МДК 04.02**

**Угрозы надёжности программного обеспечения: Какие основные виды угроз могут нарушить стабильную работу системы? Объясните механизмы их возникновения и влияния на функциональность. Приведите примеры реальных случаев сбоев программного обеспечения, описав их причины, последствия и меры, принятые для предотвращения подобных инцидентов в будущем.**

**Ответ:**

**Угрозы надёжности программного обеспечения** — это события или факторы, которые приводят к сбоям, частичной или полной недоступности системы и возникновению критических ситуаций.

**Основные угрозы надёжности:**

1. **Отказы компонентов.** Поломка одного из элементов системы может нарушить работу всей системы.
   * **Пример:** Сбой в распределённой системе **Amazon S3** привёл к глобальной недоступности файлов, хранящихся в облаке.
2. **Ошибки в обработке данных.** Некорректная обработка входных данных вызывает сбои и ошибки.
   * **Пример:** В платёжной системе **PayPal** неверная обработка ввода привела к некорректным транзакциям.
3. **Отсутствие резервирования.** При отсутствии резервных компонентов сбой одного элемента делает систему неработоспособной.
   * **Пример:** Поломка сервера в платёжной системе **Visa** в 2018 году привела к недоступности услуг по всей Европе.
4. **Кибератаки и эксплуатация уязвимостей.** Злоумышленники могут использовать уязвимости системы для её нарушения.
   * **Пример:** Атака **WannaCry** в 2017 году использовала уязвимость в Windows для распространения вируса-шифровальщика.
5. **Перегрузка системы.** Резкий рост числа пользователей или запросов может привести к сбою.
   * **Пример:** В октябре 2021 года сбой в **Facebook** вызвал глобальную недоступность сервисов на 6 часов, что повлияло на миллиарды пользователей.

**Реальные примеры нарушений надёжности:**

1. **Сбой Facebook (2021):**
   * **Причина:** Ошибка в маршрутизации BGP привела к «отключению» Facebook от Интернета.
   * **Последствия:** Миллионы пользователей потеряли доступ к сервисам Facebook, WhatsApp и Instagram.
   * **Вывод:** Нужны резервные пути маршрутизации и автоматическое восстановление конфигурации.
2. **Взрыв ракеты Ariane 5 (1996):**
   * **Причина:** Программное обеспечение некорректно обработало данные из-за переполнения памяти.
   * **Последствия:** Ущерб составил $500 млн, утрачены научные спутники.
   * **Вывод:** Тестирование ПО в условиях, максимально приближенных к реальным, необходимо.
3. **Сбой Boeing 737 MAX (2018-2019):**
   * **Причина:** Система MCAS неправильно интерпретировала данные от датчика угла атаки.
   * **Последствия:** Крушение самолётов и гибель 346 человек, многомиллиардные убытки компании Boeing.
   * **Вывод:** Для критически важных систем необходимы резервирование данных и многоуровневая проверка.

Анализ угроз надёжности программного обеспечения и изучение реальных примеров сбоев помогает разработчикам разрабатывать устойчивые системы. Устранение отговоров компонентов, резервирование и постоянный мониторинг минимизируют риски возникновения ошибок

**Вопрос 7 МДК 04.02**

**Какие ключевые методы используются для предотвращения угроз надёжности программного обеспечения? Как их применение варьируется на этапах проектирования, тестирования и эксплуатации? Приведите примеры из российского рынка программных продуктов, иллюстрирующие успешное внедрение таких методов.**

**Ответ:**

Методы предотвращения угроз надёжности программного обеспечения охватывают **три основных этапа жизненного цикла разработки и эксплуатации ПО**: проектирование, тестирование и эксплуатация.

1. **Этап проектирования:**
   * На этом этапе формируется архитектура системы с учётом устойчивости и отказоустойчивости.
   * **Методы:**
     + **Устойчивость архитектуры**: Использование распределённых систем, которые минимизируют влияние отказов. *Пример:* «Т Банк» применяет микросервисную архитектуру. Даже если один микросервис выходит из строя, другие продолжают работу.
     + **Разработка с учётом отказоустойчивости**: Проектирование систем с возможностью восстановления после сбоев. *Пример:* В «Сбере» реализовано резервное хранение данных, что исключает их потерю при сбое.
     + **Валидация данных:** Проверка входных данных на корректность. *Пример:* доподлинно это конечно неизвестно, но если рассуждать логически, то можно сделать вывод о том, что в интернет-магазине Wildberries реализована проверка данных пользователя на этапе оформления заказа, чтобы избежать ошибок в системе логистики (например, неверного формата почтового индекса).
2. **Этап тестирования:**
   * На этом этапе системы проходят проверку надёжности и отказоустойчивости.
   * **Методы:**
     + **Автоматизированное тестирование**: Использование инструментов для автоматической проверки функциональности. *Пример:* Компания «Яндекс» применяет собственные системы для тестирования поисковых и облачных сервисов.
     + **Нагрузочное тестирование**: Проверка поведения системы в условиях высокой нагрузки. *Пример:* доподлинно это конечно неизвестно, но если рассуждать логически, то можно сделать вывод о том, что Wildberries проводит стресс-тестирование перед акциями, такими как «чёрная пятница».
     + **Тестирование на отказоустойчивость**: Моделирование сбоев для проверки реакции системы. *Пример:* доподлинно это конечно неизвестно, но если рассуждать логически, то можно сделать вывод о том, что в «Сбере» применяется Chaos Engineering для выявления слабых мест в архитектуре (подход к тестированию программных систем, при котором преднамеренно создаются сбои и ошибки в рабочей среде, чтобы проверить устойчивость системы к нестандартным ситуациям. Цель метода — выявить слабые места в архитектуре, обеспечить отказоустойчивость и подготовить систему к возможным реальным сбоям).
     + **Тестирование безопасности**: Поиск уязвимостей, которые могут привести к сбоям. *Пример:* доподлинно это конечно неизвестно, но если рассуждать логически, то можно сделать вывод о том, что Госуслуги проводят регулярное пентестирование для проверки безопасности платформы.
3. **Этап эксплуатации:**
   * На этапе эксплуатации осуществляется мониторинг системы и реакция на потенциальные угрозы.
   * **Методы:**
     + **Мониторинг состояния систем**: Отслеживание работы серверов, баз данных и приложений. *Пример:* доподлинно это конечно неизвестно, но если рассуждать логически, то можно сделать вывод о том, что «1С» использует системы мониторинга для своих облачных решений (сервис 1С:Fresh).
     + **Регулярные обновления**: Обновление устаревших компонентов для устранения уязвимостей. *Пример:* Обновления платформы MSOffice устраняют критические ошибки и улучшают стабильность (MSOffice 2007 – 3 Service Pack).
     + **Сценарии реагирования на сбои**: Разработка плана действий в случае отказов. *Пример:* Мосэнерго использует автоматизированную систему для быстрого восстановления после сбоя.

**Вывод:** методы предотвращения угроз надёжности применяются на всех этапах разработки и эксплуатации ПО. Примеры из российского рынка, такие как «Тинькофф Банк», «Сбер», Wildberries и Госуслуги, демонстрируют, что сочетание устойчивой архитектуры, тщательного тестирования и постоянного мониторинга позволяет обеспечить высокую надёжность программного обеспечения.

**Вопрос 8 МДК 04.02**

**Как использование методов резервирования и обеспечения отказоустойчивости способствует предотвращению угроз надёжности в программных системах? Какие виды резервирования существуют, и как они применяются на практике? Приведите примеры успешной реализации этих подходов в программных продуктах.**

**Ответ:**

**Резервирование и отказоустойчивость** — это ключевые методы обеспечения надёжности систем. Они помогают минимизировать влияние сбоев на работу программного обеспечения и обеспечить его непрерывность.

**Методы резервирования и отказоустойчивости:**

1. **Резервирование**

Резервирование заключается в создании дублирующих систем или компонентов, которые могут заменить вышедшие из строя элементы.

* + **Пример:** Крупные Российские компании использует резервные дата-центры. Если один центр выходит из строя, другой автоматически подхватывает нагрузку.

1. **Географическая распределённость**

Разделение ресурсов между разными регионами помогает минимизировать влияние локальных сбоев (например, отказ в одной локации).

* + **Пример:** DNS-хостинг Яндекс размещает свои серверы в разных частях России, обеспечивая высокую доступность.

1. **Балансировка нагрузки**

Балансировка распределяет трафик между несколькими серверами, предотвращая их перегрузку.

* + **Пример:** Применение CDN серверов видеохостингом RUTUBE помогает распределить нагрузку на сети передачи данных и серверы компании для различных регионов страны.

1. **Резервирование данных**

Дублирование важных данных для защиты от их потери.

* + **Пример:** В Российских банках используются системы резервного хранения данных, которые обеспечивают восстановление информации при сбое (Сбербанк к примеру).

1. **Резервирование компонентов**

Использование дублирующих аппаратных компонентов, таких как диски и серверы.

* + **Пример:** ЦОД 1С (облачные сервисы) используют резервные источники питания и дублирующие компоненты для защиты от отказов оборудования.

**Примеры из российского рынка:**

1. **Госуслуги:** Для обеспечения доступности портала в условиях высокой нагрузки используется балансировка нагрузки и резервные серверы.
2. **Wildberries:** Перед крупными акциями проводятся нагрузочные тесты и настраивается балансировка нагрузки, что помогает избежать перегрузок.
3. **СберБанк:** Внедрение географически распределённых дата-центров и микросервисная архитектура позволяют продолжать работу даже при частичных отказах.
4. **RUTUBE:** Применение CDN серверов помогает распределить нагрузку на сети передачи данных и серверы компании для различных регионов страны.

**Вывод:** Применение методов резервирования и отказоустойчивости позволяет минимизировать вероятность сбоев и обеспечить стабильную работу программных систем. Опыт российских компаний, таких как «Сбер», Wildberries, Госуслуги и RUTUBE показывает, что использование резервных систем и распределения нагрузки является эффективным способом предотвращения угроз надёжности.

**Вопрос 9 МДК 04.02**

**Избыточность как метод повышения надёжности. Какие типы избыточности существуют? Опишите каждый из них с примерами применения и объясните, как они влияют на устойчивость системы.**

===============================================================================

**Ответ:**

**Избыточность** — это метод повышения надёжности, предполагающий добавление дополнительных ресурсов (времени, данных или программных компонентов) для обеспечения устойчивости системы к сбоям. Этот подход применяется для минимизации ошибок, повышения стабильности работы и восстановления системы после отказов.

**Типы избыточности:**

1. **Временная избыточность:**
   * Предполагает повторное выполнение операций для компенсации временных сбоев.
   * **Примеры:**
     + **Повторные запросы:** В случае временного сбоя соединения система повторяет запрос.*Пример:* Онлайн-магазины, такие как Wildberries и Ozon восстанавливают транзакции при временных ошибках сети.
     + **Тайм-ауты:** Если операция занимает больше заданного времени, она перезапускается. *Пример:* В сетевом протоколе TCP тайм-ауты используются для повторной передачи данных.
   * **Влияние:** Устойчивость к временным сбоям, но увеличивается время выполнения операций.
2. **Информационная избыточность:**
   * Добавление дополнительных данных для обнаружения, исправления ошибок или восстановления информации.
   * **Примеры:**
     + **Контрольные суммы (CRC):** Помогают выявить ошибки при передаче данных. *Пример:* Используются в сетевых протоколах и файловых системах.
     + **Хэмминг-коды:** Позволяют обнаруживать и исправлять ошибки при передаче данных. *Пример:* Применяются в спутниковой связи.
     + **RAID-массивы:** Дублируют данные для восстановления в случае утраты. *Пример:* Файловый сервер СТОТиС используют RAID из нескольких дисковых накопителей для защиты информации.
   * **Влияние:** Высокая точность и надёжность передачи данных, но увеличивается объём хранимой информации.
3. **Программная избыточность:**
   * Создание дополнительных программных компонентов для повышения устойчивости.
   * **Примеры:**
     + **Независимое дублирование модулей:** Создаются два модуля для выполнения одной функции. *Пример:* В системах управления железнодорожным движением дублируются модули для проверки сигналов. В авиации и на морском транспорте дублируются системы управления.
     + **Система голосования:** Несколько модулей выполняют задачу, и результат определяется на основе большинства. *Пример:* Применяется в энергетических системах для повышения точности решений.
   * **Влияние:** Повышение устойчивости к программным сбоям, но увеличиваются затраты на разработку.

**Вывод:** Каждый тип избыточности имеет свои преимущества и ограничения. Временная избыточность компенсирует временные сбои, информационная обеспечивает точность передачи данных, а программная устойчивость к внутренним сбоям. Их применение зависит от специфики системы.

**Вопрос 10 МДК 04.02**

**Реализация временной избыточности на практике. Опишите принципы, преимущества и ограничения временной избыточности. Приведите примеры из реальных систем, включая сетевые протоколы и пользовательские приложения.**

**Ответ:**

**Временная избыточность** — это метод повышения надёжности, при котором одна и та же операция выполняется несколько раз, чтобы компенсировать временные сбои. Этот подход применяется в системах, где временные ошибки могут быть исправлены без вмешательства пользователя.

**Принципы временной избыточности:**

1. **Повторные попытки (Retry):** Если операция завершилась с ошибкой, система автоматически повторяет попытку. *Пример:* В банковских приложениях, повторные запросы позволяют завершить транзакцию при нестабильности сети.
2. **Тайм-ауты:** Если операция занимает больше допустимого времени, она перезапускается. *Пример:* В протоколе TCP тайм-ауты используются для повторной передачи данных при потере пакетов.
3. **Резервные операции:** В случае сбоя выполняются резервные операции для восстановления работоспособности. *Пример:* Системы обработки платежей сохраняют транзакции и повторяют их при восстановлении соединения.

**Преимущества временной избыточности:**

* Повышает устойчивость системы к временным сбоям, вызванным нестабильной сетью или короткими отказами оборудования.
* Простота реализации: многие повторные операции требуют минимальных изменений в архитектуре.

**Ограничения:**

1. **Увеличение времени выполнения операций:** Многократные попытки могут замедлить обработку.
2. **Неэффективность при системных сбоях:** Если ошибка вызвана системной проблемой, повторные попытки не помогут.
3. **Дополнительная нагрузка:** Частые повторные запросы могут перегрузить систему, особенно в случае массовых сбоев.

**Реальные примеры временной избыточности:**

1. **Сетевые протоколы (TCP):** Тайм-ауты и повторные отправки данных гарантируют их доставку при нестабильных соединениях.
2. **Онлайн-банкинг:** Банковские приложения используют повторные попытки соединения для завершения операций.
3. **Облачные сервисы:** Сервисы вроде Google Drive повторно пытаются загрузить файл при разрыве соединения.

Программа с временной избыточностью на C++:

1. Программа без методов повышения надежности

Эта версия программы принимает число от пользователя. Если ввод некорректный, программа завершается с ошибкой.

#include <iostream>

int main() {

int number;

std::cout << "Введите целое число: ";

std::cin >> number;

if (std::cin.fail()) {

std::cerr << "Ошибка ввода. Завершение программы.\n";

return 1;

}

std::cout << "Вы ввели: " << number << std::endl;

return 0;

}

Программа уязвима к ошибкам ввода. Если пользователь вводит текст вместо числа, программа завершает работу.

2. Программа с использованием временной избыточности

Эта версия программы повторяет запрос ввода, если пользователь вводит некорректные данные.

#include <iostream>

#include <limits>

int main()

{

int number;

// Количество попыток

int attempts = 3;

system("chcp 65001");

system("cls");

while (attempts > 0)

{

std::cout << "Введите целое число (осталось попыток: " << attempts << "): ";

std::cin >> number;

if (!std::cin.fail())

{

std::cout << "Вы ввели: " << number << std::endl;

return 0;

}

else

{

std::cerr << "Ошибка ввода. Попробуйте ещё раз.\n";

// Сброс флага ошибки

std::cin.clear();

// Очистка ввода

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

--attempts;

}

}

std::cerr << "Вы исчерпали все попытки. Завершение программы.\n";

return 1;

}

Программа допускает до трёх попыток ввода.

Если ввод остаётся некорректным, программа сообщает об исчерпании попыток и завершает работу.

**Вывод:** Временная избыточность эффективна для устранения временных сбоев, что делает её важным инструментом в сетевых протоколах, онлайн-сервисах и других критически важных системах. Однако её применение требует осторожности, чтобы избежать перегрузки системы.

**Вопрос 11 МДК 04.02**

**Первичные ошибки в программном обеспечении. Какие типы первичных ошибок существуют, как они влияют на работу программы, и почему их своевременное обнаружение так важно? Подкрепите свой ответ примерами, демонстрирующими реальные последствия таких ошибок.**

**Ответ:**

**Первичные ошибки** — это ошибки, которые возникают непосредственно из-за действий программиста, таких как неверная логика алгоритма, синтаксические ошибки или некорректное использование API. Эти ошибки чаще всего обнаруживаются на ранних этапах тестирования, так как их влияние напрямую отражается на функциональности программы.

**Типичные примеры первичных ошибок:**

1. **Логические ошибки:**

Неверное условие в операторе if.

*Пример: if (x = 5) // Присваивание вместо сравнения*

Из-за этой ошибки программа всегда считает условие истинным.

1. **Синтаксические ошибки:**

Пропуск точки с запятой.

*Пример: int x = 5*

Код не компилируется из-за отсутствия ;.

1. **Ошибки в алгоритмах:**

Неправильный расчёт среднего значения массива.

*Пример:* Если пропустить один из элементов массива, результат будет некорректным.

**Влияние первичных ошибок на программу:**

* **Неправильные результаты:**

Например, ошибка в расчёте суммы чисел приводит к тому, что пользователю возвращается неверное значение.

* **Критические сбои:**

Если программа делит на ноль без проверки, это приведёт к аварийному завершению.

*Пример:*

int divide(int a, int b) {

return a / b;

}

Вызов divide(10, 0) вызовет ошибку времени выполнения.

**Почему важно вовремя выявлять первичные ошибки?**

* Первичные ошибки легко обнаружить на ранних этапах тестирования, если использовать:
  + Функциональное тестирование.
  + Инструменты статического анализа.
* Чем раньше будет исправлена ошибка, тем меньше её влияние на последующие этапы разработки.

**Пример из практики:**

* В известном случае с ракетой Ariane 5 (1996 год) логическая ошибка в программном обеспечении привела к разрушению ракеты из-за некорректного преобразования данных. Это подчёркивает важность тщательного тестирования и анализа логики.

**Вопрос 12 МДК 04.02**

**Вторичные ошибки в программном обеспечении. Какова связь между первичными и вторичными ошибками, какие основные проявления вторичных ошибок существуют? Приведите примеры вторичных ошибок и объясните, почему их сложнее выявить и устранить.**

**Ответ:**

**Вторичные ошибки** — это побочные эффекты первичных ошибок. Они проявляются в других модулях программы или на этапе взаимодействия с внешними системами. Вторичные ошибки сложнее выявить, так как их источник может находиться в другом участке кода.

**Типичные примеры вторичных ошибок:**

1. **Некорректные данные в других модулях:**

Если первичная ошибка генерирует неправильный результат, он передаётся в другой модуль и вызывает сбой.

*Пример:*

int calculatePrice(int basePrice, int discount) {

return basePrice - discount; // discount > basePrice

}

Передача отрицательной цены в модуль оформления заказа вызывает сбой.

1. **Сбои во взаимодействии с внешними системами:**

Ошибка в SQL-запросе приводит к сбою работы базы данных.

*Пример:*

std::string getUserData(int userId)

{

return "SELECT \* FROM users WHERE id = " + std::to\_string(userId);

}

Если кавычки не добавлены, запрос будет некорректным.

1. **Потеря данных:**

Первичная ошибка при обрезке строки приводит к утрате части данных.

*Пример:*

void saveData(const std::string& data)

{

std::cout << data.substr(0, 10);

}

Если длина строки меньше 10 символов, данные теряются.

1. **Необратимые сбои:**

Ошибка при удалении файла может привести к утрате критически важных данных.

*Пример:*

void deleteFile(const std::string& filePath)

{

// Ошибка: программа удаляет файл без проверки

}

**Почему вторичные ошибки сложнее выявить?**

1. Они проявляются только при взаимодействии модулей, поэтому их сложно обнаружить на этапе модульного тестирования.
2. Требуют интеграционного тестирования или анализа всей системы.
3. Иногда проявляются только в определённых сценариях, что делает их трудно воспроизводимыми.

**Примеры проявлений вторичных ошибок:**

**Сбои в сложных системах:** Ошибка в обработке данных в одном модуле нарушает работу других. *Пример:* В случае с ракетой Ariane 5 первичная ошибка передала некорректные данные в алгоритм управления, что привело к катастрофе.

**Почему важно устранять первичные ошибки?** Устранение первичной ошибки автоматически исправляет связанные с ней вторичные сбои. Для этого важно:

1. Использовать интеграционное тестирование.
2. Применять автоматизированные тесты для проверки взаимодействия модулей.
3. Проводить логирование всех этапов выполнения программы.

**Вывод:** Вторичные ошибки — это результат первичных, и их устранение требует комплексного подхода. Понимание взаимосвязи между ними помогает разработчикам предотвращать критические сбои в системах.

**Вопрос 13 МДК 04.02**

**Модули адаптации в программных системах. Каковы их основные функции, и в каких ситуациях их разработка становится целесообразной? Приведите примеры, демонстрирующие их применение на практике.**

**Ответ:**

Модули адаптации — это компоненты программных систем, которые позволяют ПО автоматически подстраиваться под изменения в окружающей среде или системе, обеспечивая её работоспособность и эффективность. Такие модули выполняют роль регуляторов, внося изменения в конфигурацию, параметры или функционал программы для достижения оптимальной работы.

Основные функции модулей адаптации:

1. Управление ресурсами:
   * Перераспределение процессорного времени, памяти и каналов связи.
   * *Пример:* В облачных системах модули адаптации перераспределяют ресурсы между виртуальными машинами в зависимости от их текущей нагрузки.
2. Изменение конфигурации:
   * Настройка параметров системы без её перезапуска.
   * *Пример:* Система видеоконференций автоматически снижает качество видео при низкой скорости интернет-соединения.
3. Предотвращение отказов:
   * Выявление и устранение проблем до того, как они вызовут сбой.
   * *Пример:* Сервер базы данных автоматически переключается на резервный экземпляр при сбое основного.
4. Интеграция новых модулей:
   * Подключение новых функций без полной модернизации системы.
   * *Пример:* Умный дом может подключить новый датчик температуры через модуль адаптации.
5. Анализ состояния системы:
   * Постоянный мониторинг и прогнозирование возможных сбоев.
   * *Пример:* Система снижает нагрузку на оборудование, прогнозируя перегрев по текущим показателям температуры.

Ситуации, где разработка модулей адаптации целесообразна:

1. Нестабильная внешняя среда:
   * ПО должно адаптироваться к переменным нагрузкам, например, в онлайн-магазинах во время распродаж.
2. Высокая стоимость простоев:
   * Например, в банковских системах модули адаптации снижают вероятность критических сбоев.
3. Необходимость интеграции с другими системами:
   * В сложных архитектурах модули упрощают процесс взаимодействия компонентов.

Как модули адаптации улучшают производительность и устойчивость?

* Обеспечивают автоматическое перераспределение ресурсов.
* Позволяют системе оставаться функциональной даже при сбоях.
* Снижают затраты на обслуживание за счёт автоматизации.

**Вопрос 14 МДК 04.02**

**Принципы целесообразности разработки модулей адаптации. Какие критерии учитываются при принятии решения о разработке таких модулей, и почему оправданность затрат и стабильность среды являются ключевыми? Приведите примеры, где использование модулей адаптации приносит практическую выгоду.**

===============================================================================

**Ответ:**

Принципы целесообразности разработки модулей адаптации помогают определить, когда внедрение таких модулей оправдано с точки зрения затрат, времени и ожидаемой эффективности.

**Основные критерии целесообразности:**

1. **Технические критерии:**
   * **Сложность системы:** Чем сложнее система, тем выше потребность в адаптации. *Пример:* В распределённой системе умного дома модули адаптации необходимы для интеграции новых устройств.
   * **Стабильность внешней среды:** Если система работает в условиях переменной нагрузки, адаптация критически важна. *Пример:* Облачные платформы, где ресурсы перераспределяются в зависимости от времени суток.
   * **Совместимость:** Если система должна взаимодействовать с другими компонентами, адаптивные модули упрощают процесс. *Пример:* Интеграция новых платёжных систем в интернет-магазине.
2. **Экономические критерии:**
   * **Оправданность затрат:** Стоимость разработки должна окупаться за счёт повышения эффективности. *Пример:* Модуль для оптимизации серверной инфраструктуры может экономить до 30% бюджета на хостинг.
   * **Потенциальный ущерб от отказа:** В критических системах, где простой может привести к убыткам, адаптация особенно важна. *Пример:* Банковская система внедряет модули, чтобы избежать финансовых потерь.
3. **Операционные критерии:**
   * **Частота изменений:** Если система часто обновляется, адаптивные модули снижают риск ошибок. *Пример:* Приложение для доставки обновляет алгоритмы маршрутизации ежедневно.
   * **Уровень автоматизации:** Для систем с большим количеством ручных операций адаптация упрощает процессы. *Пример:* Складская система автоматически пересчитывает остатки.

**Вопрос 15 МДК 04.02**

**Классификация вредоносных программ. Какие основные типы вредоносного ПО существуют, как они отличаются друг от друга по функциональности и способу воздействия на систему? Приведите примеры для каждого типа и объясните, как они влияют на пользователей и инфраструктуру.**

**Ответ:**

**Вредоносное программное обеспечение (malware)** — это программы или скрипты, специально созданные для выполнения вредоносных действий, таких как кража данных, нарушение работы систем или использование их ресурсов без согласия пользователя.

**Основные типы вредоносного ПО:**

1. **Вирусы:**
   * Внедряются в другие файлы и активируются при их выполнении. Требуют действий пользователя для распространения.
   * *Пример:* ILOVEYOU (2000 год) — вирус распространялся через электронные письма и заражал устройства после открытия вложений.
2. **Черви:**
   * Распространяются автономно, используя уязвимости в сети.
   * *Пример:* WannaCry (2017 год) — шифровал данные пользователей и требовал выкуп.
3. **Трояны:**
   * Маскируются под полезные программы, но выполняют вредоносные действия после установки.
   * *Пример:* Emotet — троян, который устанавливает другие вредоносные программы.
4. **Шпионское ПО (Spyware):**
   * Собирать личные данные, такие как пароли или банковские реквизиты.
   * *Пример:* Pegasus — шпионская программа, способная записывать звонки и сообщения.
5. **Рекламное ПО (Adware):**
   * Показывает навязчивую рекламу и отслеживает поведение пользователя.
   * *Пример:* Заражённые браузерные расширения.
6. **Программы-вымогатели (Ransomware):**
   * Шифруют данные и требуют выкуп за их восстановление.
   * *Пример:* LockBit — активно атакует корпорации.
7. **Ботнеты:**
   * Сети заражённых устройств для проведения DDoS-атак или рассылки спама.
   * Владельцы устройств часто не знают о заражении.
8. **Руткиты:**
   * Скрывают присутствие вредоносного ПО в системе, делая его обнаружение сложным.
9. **Кейлоггеры:**
   * Записывают нажатия клавиш для кражи паролей или другой конфиденциальной информации.

**Влияние на системы и пользователей:**

* Нарушение работы систем.
* Кража личных данных и финансовых ресурсов.
* Потеря файлов или доступ к устройству.

Эффективная защита от вредоносного ПО требует понимания его типов и методов воздействия.

**Вопрос 16 МДК 04.02**

**Методы обнаружения вредоносного ПО. Какие подходы применяются для выявления угроз, в чём их особенности и ограничения? Объясните, как работает эвристический анализ и почему использование песочниц является важным этапом обнаружения.**

**Ответ:**

**Методы обнаружения вредоносного ПО** помогают выявлять и предотвращать его активность. Они различаются по подходу к анализу угроз и имеют свои особенности.

**Основные методы:**

1. **Сигнатурный анализ:**
   * Сравнение кода с базой данных известных угроз.
   * *Особенность:* Высокая точность для известных угроз.
   * *Ограничение:* Бесполезен для новых и модифицированных вредоносных программ.
2. **Поведенческий анализ:**
   * Наблюдение за активностью программы во время выполнения.
   * *Особенность:* Выявляет подозрительное поведение, например, шифрование файлов.
   * *Ограничение:* Может вызвать ложные срабатывания.
3. **Эвристический анализ:**
   * Использует алгоритмы для поиска угроз, анализируя их структуру и поведение.
   * *Как работает:*
     + Анализирует код программы на наличие подозрительных элементов.
     + Сравнивает его с известными паттернами вредоносных программ.
   * *Преимущество:* Эффективен для новых угроз, ещё не известных базам данных.
   * *Ограничение:* Высокая вероятность ложных срабатываний.
4. **Использование песочниц (sandboxing):**
   * Запуск программ в изолированной среде для анализа их поведения.
   * *Как работает:*
     + Программа запускается в виртуальной системе.
     + Анализируется её влияние на файловую систему, сеть и память.
   * *Преимущество:* Полный контроль над процессом анализа.
   * *Ограничение:* Ресурсоёмкость и невозможность анализировать сложные угрозы, способные выявить виртуальную среду.

**Значение этих методов:** Эвристический анализ и песочницы играют ключевую роль в борьбе с новыми угрозами, позволяя предотвращать атаки до их распространения.

**Пример из практики:**

* Антивирусы, такие как Kaspersky и Dr.Web (Dr.Web vxCube - интеллектуальный интерактивный анализатор подозрительных объектов («песочница»)) , активно используют песочницы для анализа подозрительных файлов, предотвращая заражение. Эвристический анализ помогает выявлять модифицированные версии известных угроз.

**Вопрос 17 МДК 04.02**

**Классификация антивирусных программ. Какие основные виды антивирусов существуют, как они классифицируются по методам работы, типу защиты и области применения? Приведите примеры для каждого вида и объясните их особенности.**

**Ответ:**

**Антивирусные программы** — это специализированные приложения для защиты устройств от вредоносного программного обеспечения (malware). Они классифицируются по нескольким параметрам, включая методы работы, типы защиты и области применения.

**1. По методам работы:**

1. **Сигнатурные антивирусы:**
   * Сравнивают файлы с базой данных сигнатур, то есть уникальных «отпечатков» известных угроз.
   * *Пример:* Dr.Web Enterprise Security Suite, Kaspersky Endpoint Security Cloud.
   * **Особенности:** Высокая точность для известных угроз, но неэффективны против новых.
2. **Эвристические антивирусы:**
   * Анализируют структуру и поведение файлов, выявляя угрозы, которые ещё не попали в базы данных.
   * *Пример:* Kaspersky Endpoint Security Cloud.
   * **Особенности:** Эффективны против новых угроз, но могут вызывать ложные срабатывания.
3. **Поведенческие антивирусы:**
   * Отслеживают действия программ в реальном времени, блокируя подозрительную активность.
   * *Пример:* Symantec Endpoint Security.
   * **Особенности:** Хорошо работают против программ-вымогателей, но увеличивают нагрузку на систему.
4. **Облачные антивирусы:**
   * Используют удалённые серверы для анализа данных.
   * *Пример:* Kaspersky Endpoint Security Cloud Plus.
   * **Особенности:** Снижают нагрузку на устройства пользователя, но требуют стабильного подключения к интернету.

**2. По типу защиты:**

1. **Локальные антивирусы:**
   * Работают автономно на устройстве.
   * *Пример:* Windows Defender.
   * **Особенности:** Просты в использовании, но ограничены локальной базой данных.
2. **Сетевые антивирусы:**
   * Предназначены для защиты серверов и сетевой инфраструктуры.
   * *Пример:* Dr.Web Server Security Suite, Dr.Web Gateway Security Suite, Dr.Web Enterprise Security Suite, Symantec Endpoint Protection.
   * **Особенности:** Подходят для корпоративных клиентов.
3. **Гибридные антивирусы:**
   * Сочетают локальные и облачные технологии.
   * *Пример:* Kaspersky Total Security.
   * **Особенности:** Универсальны, обеспечивают высокий уровень защиты.

**3. По области применения:**

1. **Для домашних пользователей:**
   * Простые интерфейсы, базовые функции.
   * *Пример:* Dr.Web Security Space.
2. **Для корпоративных клиентов:**
   * Расширенные функции, защита серверов и сетей.
   * *Пример:* Dr.Web Enterprise Security Suite, Symantec Enterprise, Kaspersky Endpoint Security Cloud.
3. **Специализированные антивирусы:**
   * Для мобильных устройств, IoT и промышленных систем.
   * *Пример:* Dr.Web Industrial.

Классификация антивирусов позволяет выбрать подходящий продукт в зависимости от целей и задач, будь то защита домашнего устройства или корпоративной сети.

**Вопрос 18 МДК 04.02**

**Методы работы антивирусных программ. Какие основные методы обнаружения угроз используют современные антивирусы? Объясните, как работают сигнатурный, эвристический и поведенческий анализ, а также их преимущества и ограничения. Приведите примеры антивирусов, использующих эти методы.**

**Ответ:**

**Методы работы антивирусов** включают различные подходы к обнаружению угроз. Они могут сочетать несколько методов для повышения эффективности защиты.

**1. Сигнатурный анализ:**

* **Принцип работы:** Сравнивает файлы на устройстве с базой данных сигнатур — уникальных «отпечатков» вредоносных программ.
* **Пример работы:**
  + Пользователь скачивает файл.
  + Антивирус проверяет файл по базе данных.
  + Если файл совпадает с известной угрозой, он блокируется.
* **Примеры антивирусов:** Dr.Web, Kaspersky.
* **Преимущества:**
  + Высокая точность для известных угроз.
* **Недостатки:**
  + Неэффективен для новых угроз, которые ещё не попали в базу.

**2. Эвристический анализ:**

* **Принцип работы:** Исследует структуру и поведение файлов для обнаружения новых угроз.
* **Пример работы:**
  + Программа с подозрительными инструкциями блокируется, даже если её сигнатура неизвестна.
* **Примеры антивирусов:** Kaspersky, Dr.Web.
* **Преимущества:**
  + Выявляет новые и модифицированные угрозы.
* **Недостатки:**
  + Высокий риск ложных срабатываний.

**3. Поведенческий анализ:**

* **Принцип работы:** Отслеживает действия программ в реальном времени. Если программа пытается изменить системные файлы или шифровать данные, она блокируется.
* **Пример работы:**
  + Антивирус блокирует приложение, которое без разрешения меняет системные настройки.
* **Примеры антивирусов:** Kaspersky, Dr.Web.
* **Преимущества:**
  + Эффективен против программ-вымогателей.
* **Недостатки:**
  + Может создавать нагрузку на систему.

**4. Облачные технологии:**

* **Принцип работы:** Используют удалённые серверы для анализа угроз.
* **Примеры антивирусов:** Dr.Web, Kaspersky.
* **Преимущества:**
  + Минимальная нагрузка на устройство пользователя.
* **Недостатки:**
  + Требуют постоянного подключения к интернету.

Современные антивирусы, такие как Kaspersky, Dr.Web, используют комбинированные методы анализа угроз для обеспечения максимальной защиты. Каждый метод имеет свои преимущества и ограничения, что делает их совместное применение наиболее эффективным.

**Вопрос 19 МДК 04.02**

**Файрволы: задачи и основные принципы работы. Какие задачи решают файрволы и как реализуются их основные функции, такие как фильтрация пакетов, Stateful Inspection и глубокий анализ пакетов (DPI)? Приведите примеры использования каждого подхода.**

**Ответ:**

**Файрвол (брандмауэр)** — это программное или аппаратное средство защиты, которое контролирует входящий и исходящий сетевой трафик, фильтруя данные в соответствии с заданными правилами безопасности. Файрволы защищают системы от кибератак, несанкционированного доступа и нежелательного трафика.

**Основные задачи файрволов:**

1. **Контроль доступа:** Управление доступом к ресурсам сети на основе политик безопасности. *Пример:* Ограничение доступа сотрудников к корпоративным данным из публичных сетей.
2. **Фильтрация трафика:** Блокировка нежелательных или вредоносных данных. *Пример:* Запрет доступа к портам, используемым для несанкционированного сканирования.
3. **Защита от атак:** Предотвращение DDoS-атак, сканирования портов и других угроз. *Пример:* Защита от массовой отправки запросов на веб-сервер.
4. **Мониторинг:** Сбор данных для анализа безопасности и выявления подозрительной активности.

**Основные принципы работы:**

1. **Фильтрация пакетов (Packet Filtering):**
   * Проверка заголовков пакетов по IP-адресам, портам и протоколам.
   * *Пример:* Разрешить трафик только на порт 443 (HTTPS) и блокировать порт 21 (FTP).
   * **Особенности:** Простая реализация, но ограниченная защита от сложных угроз.
2. **Stateful Inspection:**
   * Анализ активных соединений для проверки, являются ли пакеты частью допустимой сессии.
   * *Пример:* Если пользователь инициировал веб-запрос, ответы от сервера автоматически разрешаются.
   * **Особенности:** Обеспечивает более высокий уровень защиты.
3. **Глубокий анализ пакетов (Deep Packet Inspection, DPI):**
   * Анализ содержимого пакетов на уровне приложений.
   * *Пример:* DPI может блокировать передачу файлов с вредоносным кодом даже через разрешённый порт.
   * **Особенности:** Высокая эффективность против сложных угроз, но требует значительных ресурсов.

Файрволы обеспечивают многоуровневую защиту сетей и приложений, применяя разные методы фильтрации и анализа данных. Комбинация технологий, таких как Stateful Inspection и DPI, позволяет минимизировать риски и защищать от сложных атак.

**Вопрос 20 МДК 04.02**

**Классификация и сравнительный анализ файрволов. Какие виды файрволов существуют по принципу работы и уровню защиты? Как они применяются в разных средах, включая домашние сети и корпоративные инфраструктуры? Обоснуйте выбор конкретного типа файрвола для примера из практики.**

**Ответ:**

**Классификация файрволов** основана на их принципах работы, уровне защиты и способе развертывания. Это позволяет адаптировать решения под конкретные потребности пользователей.

**1. По уровню защиты:**

1. **Сетевые файрволы:**
   * Фильтруют пакеты на уровне сетевого взаимодействия.
   * *Пример:* Континент-AP обеспечивает базовую защиту на уровне IP-адресов и портов.
2. **Прикладные файрволы (WAF):**
   * Защищают веб-приложения от атак на уровне HTTP/HTTPS.
   * *Пример:* PT Application Firewall защищает от SQL-инъекций.
3. **DPI файрволы:**
   * Выполняют глубокий анализ данных, включая содержимое пакетов.
   * *Пример:* UserGate.

**2. По принципу работы:**

1. **Пакетные фильтры:**
   * Проверяют только заголовки пакетов.
   * *Пример:* Континент-AP.
   * **Особенности:** Высокая производительность, но слабая защита.
2. **Сетевые экраны (Stateful Inspection):**
   * Анализируют соединения на уровне сессий.
   * *Пример:* UserGate.
   * **Особенности:** Учитывают контекст соединений, обеспечивая баланс между защитой и производительностью.
3. **DPI файрволы:**
   * Выполняют глубокий анализ содержимого пакетов.
   * *Пример:* UserGate.
   * **Особенности:** Высокая точность, но высокая нагрузка.

**3. По способу развертывания:**

1. **Программные файрволы:**
   * Устанавливаются на устройства пользователя.
   * *Пример:* Windows Defender.
   * **Особенности:** Простая настройка, подходит для домашних пользователей.
2. **Аппаратные файрволы:**
   * Используются в корпоративных сетях.
   * *Пример:* Континент-AP.
   * **Особенности:** Высокая производительность, но высокая стоимость.
3. **Облачные файрволы:**
   * Предоставляют защиту через удалённые серверы.
   * *Пример:* Результаты таких решений в России ограничены.

**Применение в разных средах:**

1. **Домашние сети:**
   * Рекомендуется использовать программные файрволы, такие как Windows Defender.
2. **Малые компании:**
   * Оптимальны аппаратные решения начального уровня, например, Континент-AP.
3. **Крупные организации:**
   * Требуются DPI файрволы, такие как UserGate, с функциями NGFW.

**Пример:** Средний бизнес использует UserGate для защиты локальной сети от сложных атак. DPI анализ помогает блокировать вредоносный трафик, а функции мониторинга упрощают управление безопасностью.

Выбор файрвола зависит от уровня защиты, требований производительности и доступного бюджета. Использование российских решений, таких как Континент-AP и UserGate, позволяет обеспечить высокий уровень безопасности, соответствующий локальным стандартам.

**Вопрос 21 МДК 04.02**

**Групповые политики в корпоративной сети. Что такое групповые политики, какова их основная структура, и какие задачи они решают? Объясните, как объекты групповой политики (GPO) используются для управления доступом и безопасности, и приведите примеры их применения.**

**Ответ:**

**Групповые политики (Group Policy)** — это механизм централизованного управления настройками операционных систем, приложений и учётных записей в корпоративных сетях на базе Windows. Они позволяют автоматизировать управление конфигурацией устройств и пользователей, что существенно упрощает администрирование.

**Основная структура групповых политик:**

1. **Объект групповой политики (GPO):** Это набор правил, определяющих настройки для пользователей и устройств. GPO содержит политики безопасности, параметры приложений и прочие настройки.
2. **Контейнеры Active Directory:** Это домены, сайты и организационные единицы (OUs), где применяются политики. Например, политики можно настроить для определённой группы пользователей.
3. **Фильтры безопасности:** Ограничивают применение политик для определённых пользователей, устройств или условий.

**Основные задачи групповых политик:**

1. **Управление доступом:**
   * Регулирование прав доступа к ресурсам, таким как файлы и устройства.
   * *Пример:* Разрешение доступа к принтерам только для сотрудников определённого отдела.
2. **Безопасность:**
   * Применение политик паролей и блокировка ненужных приложений.
   * *Пример:* Запрет подключения USB-устройств для предотвращения утечек данных.
3. **Унификация настроек:**
   * Автоматизация конфигурации рабочих станций.
   * *Пример:* Настройка рабочего стола и сети для всех сотрудников.
4. **Контроль поведения пользователей:**
   * Ограничение функционала, предотвращение несанкционированного использования ресурсов.
   * *Пример:* Блокировка доступа к определённым сайтам.

**Примеры применения:**

1. **Автоматическая установка программ:** GPO позволяет централизованно устанавливать приложения на устройства сотрудников. *Пример:* Автоматическая установка Microsoft Office для всех сотрудников офиса.
2. **Настройка обновлений Windows:** Политики обеспечивают синхронизацию обновлений через централизованный сервер.
3. **Ограничение доступа к ресурсам:** Например, политика запрещает доступ к корпоративным данным вне офиса.

Групповые политики обеспечивают эффективное управление устройствами и пользователями в корпоративных сетях, улучшая безопасность и оптимизируя процессы администрирования.

**Вопрос 22 МДК 04.02**

**Аутентификация и учётные записи в системе безопасности. Какие типы аутентификации существуют, каковы их основные принципы работы, и какие методы считаются наиболее безопасными? Объясните, как учетные записи используются для управления доступом, и приведите примеры их настройки.**

**Ответ:**

**Аутентификация** — это процесс проверки подлинности пользователя или устройства перед предоставлением доступа к системе. Она играет ключевую роль в обеспечении безопасности, предотвращая несанкционированный доступ.

**Типы аутентификации:**

1. **Однофакторная аутентификация (SFA):** Использует один фактор проверки, чаще всего логин и пароль. *Пример:* Вход на веб-сайт с использованием пароля.
   * **Недостатки:** Уязвимость перед атаками, такими как перебор паролей.
2. **Двухфакторная аутентификация (2FA):** Требует два различных фактора.
   * *Пример:* Интернет-банкинг, где пользователь вводит пароль и код из SMS.
   * **Преимущества:** Высокая устойчивость к большинству атак.
3. **Многофакторная аутентификация (MFA):** Использует три или более факторов. *Пример:* Системы, использующие пароль, токен и биометрию.
   * **Преимущества:** Максимальная безопасность.

**Методы аутентификации:**

1. **Пароли:** Наиболее распространённый метод.
   * *Недостаток:* Слабые пароли легко взламываются.
2. **Биометрия:** Использует отпечатки пальцев или сканирование лица (сетчатки глаза).
   * *Пример:* Разблокировка смартфонов.
3. **Токены:** Физические устройства или приложения, генерирующие коды.
   * *Пример:* Смарт-карты для входа в корпоративные системы.
4. **Одноразовые пароли (OTP):** Генерируются для каждой сессии, повышая уровень безопасности.

**Учетные записи в управлении доступом:**

1. **Типы учетных записей:**
   * **Локальные:** Используются на конкретном устройстве.
   * **Доменные:** Позволяют централизованное управление через Active Directory.
   * **Администраторские:** Предоставляют полный доступ к системе.
2. **Настройка учётных записей:**
   * Присвоение уникального имени и пароля.
   * Установка политик паролей (сложность, срок действия).
   * Ограничение доступа по времени или местоположению.
3. **Пример управления:** Администратор создаёт временную гостевую учётную запись с ограниченным доступом к корпоративным ресурсам.

Аутентификация и учётные записи играют ключевую роль в системе безопасности. Использование многофакторной аутентификации в сочетании с грамотным управлением учётными записями позволяет минимизировать риски и защитить корпоративные данные.

**Вопрос 23 МДК 04.02**

**Методы и этапы тестирования защиты программного обеспечения. Какие ключевые этапы включает процесс тестирования безопасности, и какие методы, такие как статический анализ, динамический анализ и пентестирование, используются для выявления уязвимостей? Приведите примеры инструментов и их применения на практике.**

**Ответ:**

**Тестирование защиты программного обеспечения** — это процесс выявления уязвимостей, которые могут быть использованы злоумышленниками. Оно включает множество этапов и методов, каждый из которых направлен на обеспечение безопасности программных продуктов.

**Ключевые этапы тестирования:**

1. **Планирование:**
   * Определение целей, областей тестирования и сценариев.
   * *Пример:* В проекте интернет-магазина планирование включает проверку форм авторизации и корзины.
2. **Сканирование уязвимостей:**
   * Использование автоматизированных инструментов, таких как Burp Suite, для поиска известных уязвимостей.
3. **Проверка вручную:**
   * Анализ кода и систем безопасности специалистами.
   * *Пример:* Проверка бизнес-логики приложения вручную для поиска уязвимостей.
4. **Отчёт:**
   * Составление отчёта с обнаруженными уязвимостями, их приоритетами и рекомендациями.

**Методы тестирования и примеры инструментов:**

1. **Статический анализ:**
   * Изучение исходного кода без выполнения программы.
   * *Инструмент:* SonarQube.
   * *Пример:* Выявление SQL-инъекций в коде с помощью анализа строки:

*std::string query = "SELECT \* FROM users WHERE username = '" + username + "';";*

1. **Динамический анализ:**
   * Тестирование программы во время её работы.
   * *Инструмент:* OWASP ZAP.
   * *Пример:* Перехват HTTP-запросов для анализа их безопасности.
2. **Пентестирование (Penetration Testing):**
   * Имитация атак злоумышленников для проверки защиты.
   * *Инструмент:* Metasploit.
   * *Пример:* Проверка на возможность обхода авторизации с использованием поддельных токенов.

**Пример практического применения:** Для проверки безопасности интернет-магазина использовался SQLmap. В результате анализа был выявлен ввод, допускающий SQL-инъекции, что позволило получить доступ к базе данных. Проблема решена внедрением параметризованных запросов.

Методы тестирования безопасности, включая статический анализ, динамический анализ и пентестирование, позволяют выявлять уязвимости на различных этапах работы программного обеспечения. Это значительно повышает уровень защиты конечного продукта.

**Вопрос 24 МДК 04.02**

**SQL-инъекции как тип уязвимости. Как работают SQL-инъекции, какие их виды существуют, и какие меры можно принять для предотвращения этого типа атак? Приведите примеры, демонстрирующие механизм их работы и методы защиты.**

**Ответ:**

**SQL-инъекции (SQL Injection)** — это одна из наиболее распространённых уязвимостей веб-приложений, при которой злоумышленники внедряют вредоносные SQL-запросы в пользовательский ввод, чтобы манипулировать базой данных.

**Как работают SQL-инъекции:**

1. **Механизм взаимодействия:** Веб-приложения используют SQL-запросы для взаимодействия с базой данных. *Пример запроса:*

*SELECT \* FROM users WHERE username = 'user1' AND password = 'pass123';*

**Манипуляция запросом:** Злоумышленник вводит данные, изменяющие логику запроса. Пример вредоносного ввода:

***username: ' OR '1'='1***

***password: anything***

Итоговый запрос:

***SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = 'anything';***

**Типы SQL-инъекций:**

1. **Классическая SQL-инъекция:** Прямое добавление вредоносного кода. *Пример:*

***' UNION SELECT username, password FROM users; --***

1. **Слепая SQL-инъекция (Blind SQL Injection):** Проверка через логические операции. *Пример:*

***' AND 1=1; --***

1. **Временная SQL-инъекция:** Использование функций задержки, таких как SLEEP(). *Пример:*

*' OR IF(1=1, SLEEP(5), 0); --*

1. **Ошибка на основе SQL-инъекции:** Злоумышленник вызывает ошибки для получения данных. *Пример:*

*' UNION SELECT NULL, version(); --*

**P.S.** Апостроф “ ' ”Используется для завершения строки в SQL-запросе. В SQL строки заключаются в одинарные кавычки (например, 'username'). Если пользовательский ввод вставляется в запрос, вредоносный код добавляет апостроф, чтобы выйти из строки и начать писать SQL-команду.

**-- (два дефиса):** Означает начало комментария в SQL. Всё, что написано после --, игнорируется сервером базы данных. Это используется для "отключения" оставшейся части исходного запроса, предотвращая его выполнение. Удачи на экзамене, и это, не тупите 😊

**Меры предотвращения SQL-инъекций:**

1. **Параметризованные запросы:** Все параметры передаются отдельно от SQL-кода.
2. **Валидация ввода:** Проверка длины, формата и допустимых символов.
3. **Ограничение прав доступа:** Создание учётных записей базы данных с минимальными привилегиями.
4. **Использование веб-фаерволов (WAF):** Они фильтруют вредоносные запросы на уровне сети.

SQL-инъекции представляют серьёзную угрозу для безопасности баз данных. Понимание их механизма и применение защитных мер, таких как параметризованные запросы, помогает снизить риск атак и защитить системы.

**Вопрос 25 МДК 04.02**

**Шифрование данных: Какие задачи решает шифрование и какие основные типы алгоритмов существуют? Объясните различия между симметричным, асимметричным и гибридным шифрованием, их преимущества и недостатки. Приведите примеры алгоритмов для каждого типа и области их применения.**

**Ответ:**

**Шифрование** — это процесс преобразования данных в зашифрованный формат (шифротекст), который может быть прочитан только авторизованной стороной. Шифрование защищает данные от несанкционированного доступа.

**Основные задачи шифрования:**

1. **Конфиденциальность:** Защита данных от доступа третьих лиц. *Пример:* Передача личной информации через интернет.
2. **Целостность:** Гарантия отсутствия изменений в данных. *Пример:* Проверка файла на изменения при передаче.
3. **Аутентификация:** Подтверждение подлинности отправителя. *Пример:* Электронная подпись.
4. **Неотказуемость:** Исключение возможности отрицания факта отправки данных.

**Типы алгоритмов шифрования:**

1. **Симметричное шифрование:**
   * **Принцип работы:** Один ключ используется для шифрования и расшифровки.
   * **Примеры алгоритмов:** AES (Advanced Encryption Standard), **ГОСТ 28147-89 (отменен, заменён на** ГОСТ 34.12-2018. ГОСТ 34.13-2018)
   * **Применение:** VPN, шифрование файлов.
   * **Преимущества:** Высокая скорость.
   * **Недостатки:** Требует защищённого обмена ключами.
2. **Асимметричное шифрование:**
   * **Принцип работы:** Используются два ключа: открытый для шифрования и закрытый для расшифровки.
   * **Примеры алгоритмов:** RSA, ГОСТ Р 34.10-2012.
   * **Применение:** Цифровые подписи, шифрование API-ключей.
   * **Преимущества:** Не требует передачи секретного ключа.
   * **Недостатки:** Медленнее симметричного шифрования.
3. **Гибридное шифрование:**
   * **Принцип работы:** Сочетает симметричное и асимметричное шифрование, где асимметричное шифрование используется для обмена симметричными ключами.
   * **Пример:** Протокол TLS/SSL.
   * **Преимущества:** Высокая скорость и надёжность.
   * **Недостатки:** Сложность реализации.

Каждый тип шифрования имеет свои преимущества и недостатки, что делает их подходящими для разных задач. Например, симметричное шифрование используется в VPN, асимметричное — для цифровых подписей, а гибридное — в современных протоколах связи.