ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ»

Институт информатики, математики и робототехники
Кафедра математического и компьютерного моделирования

Лабораторная работа №12: JSON-YAML-XML

ОБУЧАЮЩЕГОСЯ
4 курса группы ПИ-4ИВТ221Б

Санникова Михаила Александровича

Уровень высшего образования:	высшее образование – бакалавриат
Направление подготовки (специальность)	09.03.03 “Прикладная информатика”
Направленность (профиль) программы	Информационные и вычислительные технологии
Дата выполнения	25.11.2025

УФА – 2025

Постановка задачи:

Необходимо разработать приложение, работающее с документами XML. Приложение должно уметь сохранять и загружать данные из форматов JSON, YAML и XML.

Этап 1: Создание XML файла с вложенной структурой
Создайте XML файл, моделирующий систему авиарейсов
Этап 2: Чтение и анализ XML файла
Напишите программу на Python, которая: □

- Читает XML файл с использованием SAX парсера
- Извлекает данные из сложных вложенных структур
- Анализирует иерархию точек распределения
- Рассчитывает метрики эффективности (загрузка мощностей, температурный режим)
- Строит карту зависимостей между элементами системы

Этап 3: Расширенная фильтрация и валидация
Этап 4: Трансформация и генерация отчетов
Разработайте систему преобразования данных, которая:

- Генерирует различные XML форматы
- Создает XSLT-трансформации для визуализации данных
- Формирует конфигурационные файлы для системы мониторинга
- Экспортирует данные в другие форматы (JSON, YAML, CSV) с сохранением структуры
- Создает схемы XSD для валидации входящих данных

Этап 5: Обработка ошибок и безопасность
Реализуйте систему обработки ошибок и обеспечения безопасности: □

- Валидация XML против XSD схемы
- Обработка XML инъекций и защита от внешних вмешательств
- Проверка целостности ссылок между элементами
- Обработка некорректных структур пространства имён
- Восстановление после ошибок парсинга с сохранением контекста

Практическая часть:

Листинг кода с комментариями

```
import xml.sax
import json
import yaml
import csv
import os
from datetime import datetime
from xml.sax import handler, parse
import xml.etree.ElementTree as ET
from typing import Dict, List, Any



# ============================================================================
# КЛАСС ДЛЯ SAX ПАРСЕРА (ЭТАП 2)
# ============================================================================

class AviationSAXHandler(handler.ContentHandler):
    def __init__(self):
        self.current_data = ""
        self.airports = []
        self.aircraft = []
        self.routes = []
        self.current_airport = None
        self.current_aircraft = None
        self.current_route = None
        self.hierarchy_map = {}

    def startElement(self, name, attrs):
        self.current_data = name
        if name == "hub" or name == "regional_airport" or name == "local_airport":
            self.current_airport = {
                'id': attrs.get('id', ''),
                'type': name,
                'region': attrs.get('region', ''),
                'parent': attrs.get('parent_hub') or attrs.get('parent_airport', '')
            }
        elif name == "aircraft":
            self.current_aircraft = {
                'id': attrs.get('id', ''),
                'type': attrs.get('type', ''),
                'base_airport': attrs.get('base_airport', '')
            }
        elif name == "route":
            self.current_route = {
                'id': attrs.get('id', '')
            }

    def characters(self, content):
        if self.current_airport and self.current_data in ['name', 'location', 'terminal_capacity']:
            if self.current_data not in self.current_airport:
                self.current_airport[self.current_data] = content.strip()
            else:
                self.current_airport[self.current_data] += content.strip()

    def endElement(self, name):
        if name == "hub" or name == "regional_airport" or name == "local_airport":
            if self.current_airport:
```

```
            self.airports.append(self.current_airport)
            self.current_airport = None
        elif name == "aircraft":
            if self.current_aircraft:
                self.aircraft.append(self.current_aircraft)
                self.current_aircraft = None
        elif name == "route":
            if self.current_route:
                self.routes.append(self.current_route)
                self.current_route = None
        self.current_data = ""


# ============================================================================
# КЛАСС ДЛЯ АНАЛИЗА И ФИЛЬТРАЦИИ (ЭТАП 3)
# ============================================================================

class AviationAnalyzer:
    def __init__(self, xml_file):
        self.xml_file = xml_file
        self.tree = ET.parse(xml_file)
        self.root = self.tree.getroot()

    def parse_with_sax(self):
        """SAX парсинг XML файла"""
        handler = AviationSAXHandler()
        parse(self.xml_file, handler)
        return handler

    def analyze_hierarchy(self):
        """Анализ иерархии аэропортов"""
        hierarchy = {
            'hubs': [],
            'regional_airports': [],
            'local_airports': []
        }

        for hub in self.root.findall('.//hub'):
            hub_data = {
                'id': hub.get('id'),
                'name': hub.find('name').text if hub.find('name') is not None else '',
                'region': hub.get('region'),
                'capacity': hub.find('terminal_capacity').text if hub.find('terminal_capacity') is not None else '0'
            }
            hierarchy['hubs'].append(hub_data)

        for regional in self.root.findall('.//regional_airports/airport'):
            regional_data = {
                'id': regional.get('id'),
                'name': regional.find('name').text if regional.find('name') is not None else '',
                'region': regional.get('region'),
                'parent': regional.get('parent_hub'),
                'capacity': regional.find('terminal_capacity').text if regional.find(
                    'terminal_capacity') is not None else '0'
            }
            hierarchy['regional_airports'].append(regional_data)

        for local in self.root.findall('.//local_airports/airport'):
```

```
            local_data = {
                'id': local.get('id'),
                'name': local.find('name').text if local.find('name') is not None else '',
                'region': local.get('region'),
                'parent': local.get('parent_airport'),
                'capacity': local.find('terminal_capacity').text if local.find('terminal_capacity') is not None else '0'
            }
            hierarchy['local_airports'].append(local_data)

        return hierarchy

    def calculate_efficiency_metrics(self):
        """Расчет метрик эффективности"""
        metrics = {
            'total_airports': 0,
            'total_capacity': 0,
            'current_passengers': 0,
            'utilization_rate': 0,
            'fuel_efficiency': 0
        }

        # Анализ аэропортов
        airports = self.root.findall('.//hub') + self.root.findall('.//regional_airports/airport') +
self.root.findall(
            './/local_airports/airport')
        metrics['total_airports'] = len(airports)

        total_capacity = 0
        total_passengers = 0

        for airport in airports:
            capacity_elem = airport.find('terminal_capacity')
            passengers_elem = airport.find('current_passengers')

            if capacity_elem is not None and capacity_elem.text:
                total_capacity += int(capacity_elem.text)
            if passengers_elem is not None and passengers_elem.text:
                total_passengers += int(passengers_elem.text)

        metrics['total_capacity'] = total_capacity
        metrics['current_passengers'] = total_passengers
        metrics['utilization_rate'] = (total_passengers / total_capacity * 100) if total_capacity > 0 else 0

        # Анализ топливной эффективности
        total_fuel_capacity = 0
        total_current_fuel = 0

        for fuel_tank in self.root.findall('.//fuel_capacity'):
            if fuel_tank.text:
                total_fuel_capacity += int(fuel_tank.text)

        for current_fuel in self.root.findall('.//current_fuel'):
            if current_fuel.text:
                total_current_fuel += int(current_fuel.text)

        metrics['fuel_efficiency'] = (total_current_fuel / total_fuel_capacity * 100) if total_fuel_capacity > 0
else 0
```

```
        return metrics

    def build_dependency_map(self):
        """Построение карты зависимостей"""
        dependency_map = {}

        # Зависимости аэропортов
        for airport in self.root.findall('.//regional_airports/airport'):
            airport_id = airport.get('id')
            parent_hub = airport.get('parent_hub')
            if airport_id and parent_hub:
                dependency_map[airport_id] = {
                    'type': 'regional_airport',
                    'parent': parent_hub,
                    'dependencies': []
                }

        for airport in self.root.findall('.//local_airports/airport'):
            airport_id = airport.get('id')
            parent_airport = airport.get('parent_airport')
            if airport_id and parent_airport:
                dependency_map[airport_id] = {
                    'type': 'local_airport',
                    'parent': parent_airport,
                    'dependencies': []
                }

        # Зависимости самолетов
        for aircraft in self.root.findall('.//aircraft'):
            aircraft_id = aircraft.get('id')
            base_airport = aircraft.get('base_airport')
            if aircraft_id and base_airport:
                dependency_map[aircraft_id] = {
                    'type': 'aircraft',
                    'parent': base_airport,
                    'dependencies': []
                }

        return dependency_map


# ============================================================================
# КЛАСС ДЛЯ ФИЛЬТРАЦИИ И ВАЛИДАЦИИ (ЭТАП 3)
# ============================================================================

class AviationFilter:
    def __init__(self, analyzer):
        self.analyzer = analyzer
        self.root = analyzer.root

    def group_by_type_and_region(self):
        """Группировка по типам и регионам"""
        grouped_data = {}

        for airport_type in ['hub', 'regional_airport', 'local_airport']:
            grouped_data[airport_type] = {}
```

```
        airports = self.root.findall(f'.//{airport_type}') if airport_type == 'hub' else self.root.findall(
            f'.//{airport_type}s/airport')

        for airport in airports:
            region = airport.get('region', 'unknown')
            if region not in grouped_data[airport_type]:
                grouped_data[airport_type][region] = []

            airport_data = {
                'id': airport.get('id'),
                'name': airport.find('name').text if airport.find('name') is not None else '',
                'location': airport.find('location').text if airport.find('location') is not None else ''
            }
            grouped_data[airport_type][region].append(airport_data)

    return grouped_data

def filter_by_complex_criteria(self, min_capacity=10000, max_fuel_usage=80, min_security=8):
    """Фильтрация по комплексным критериям"""
    filtered_airports = []

    for airport in self.root.findall('.//hub') + self.root.findall(
            './/regional_airports/airport') + self.root.findall('.//local_airports/airport'):
        capacity_elem = airport.find('terminal_capacity')
        security_elem = airport.find('security_level')

        capacity = int(capacity_elem.text) if capacity_elem is not None and capacity_elem.text else 0
        security = int(security_elem.text) if security_elem is not None and security_elem.text else 0

        # Расчет использования топлива (упрощенный)
        fuel_capacity_elem = airport.find('fuel_capacity')
        current_fuel_elem = airport.find('current_fuel')

        fuel_usage = 0
        if fuel_capacity_elem is not None and fuel_capacity_elem.text and current_fuel_elem is not None and
current_fuel_elem.text:
            fuel_capacity = int(fuel_capacity_elem.text)
            current_fuel = int(current_fuel_elem.text)
            fuel_usage = ((fuel_capacity - current_fuel) / fuel_capacity * 100) if fuel_capacity > 0 else 0

        if (capacity >= min_capacity and
                fuel_usage <= max_fuel_usage and
                security >= min_security):
            filtered_airports.append({
                'id': airport.get('id'),
                'name': airport.find('name').text if airport.find('name') is not None else '',
                'capacity': capacity,
                'fuel_usage': fuel_usage,
                'security': security
            })

    return filtered_airports

def detect_maintenance_issues(self):
    """Выявление проблем с техническим обслуживанием"""
    issues = []
```

```
        for aircraft in self.root.findall('.//aircraft'):
            maintenance_elem = aircraft.find('maintenance_schedule')
            if maintenance_elem is not None:
                next_maintenance_elem = maintenance_elem.find('next_maintenance')
                if next_maintenance_elem is not None and next_maintenance_elem.text:
                    next_maintenance = datetime.fromisoformat(next_maintenance_elem.text.replace('Z',
'+00:00'))
                    days_until_maintenance = (next_maintenance - datetime.now()).days

                    if days_until_maintenance <= 7:  # Менее недели до обслуживания
                        issues.append({
                            'aircraft_id': aircraft.get('id'),
                            'model': aircraft.find('model').text if aircraft.find('model') is not None else '',
                            'issue': 'Срочное техническое обслуживание',
                            'days_until': days_until_maintenance
                        })

        return issues

    def validate_supply_chain_integrity(self):
        """Проверка целостности цепочек поставок"""
        integrity_issues = []

        # Проверка ссылок между аэропортами
        for regional in self.root.findall('.//regional_airports/airport'):
            parent_hub = regional.get('parent_hub')
            if parent_hub and not self.root.find(f'.//hub[@id="{parent_hub}"]'):
                integrity_issues.append(
                    f"Региональный аэропорт {regional.get('id')} ссылается на несуществующий хаб
{parent_hub}")

        for local in self.root.findall('.//local_airports/airport'):
            parent_airport = local.get('parent_airport')
            if parent_airport and not (self.root.find(f'.//hub[@id="{parent_airport}"]') or self.root.find(
                    f'.//regional_airports/airport[@id="{parent_airport}"]')):
                integrity_issues.append(
                    f"Локальный аэропорт {local.get('id')} ссылается на несуществующий родительский
аэропорт {parent_airport}")

        # Проверка самолетов
        for aircraft in self.root.findall('.//aircraft'):
            base_airport = aircraft.get('base_airport')
            if base_airport and not (self.root.find(f'.//hub[@id="{base_airport}"]') or self.root.find(
                    f'.//regional_airports/airport[@id="{base_airport}"]') or self.root.find(
                    f'.//local_airports/airport[@id="{base_airport}"]')):
                integrity_issues.append(f"Самолет {aircraft.get('id')} имеет несуществующую базу
{base_airport}")

        return integrity_issues


# ============================================================================
# КЛАСС ДЛЯ ТРАНСФОРМАЦИИ И ОТЧЕТОВ (ЭТАП 4)
# ============================================================================

class AviationTransformer:
    def __init__(self, analyzer):
```

```
        self.analyzer = analyzer
        self.root = analyzer.root

    def generate_json_report(self, output_file):
        """Генерация JSON отчета"""
        report_data = {
            'metadata': {},
            'airports_summary': {},
            'aircraft_summary': {},
            'efficiency_metrics': self.analyzer.calculate_efficiency_metrics()
        }

        # Метаданные
        metadata_elem = self.root.find('metadata')
        if metadata_elem is not None:
            for child in metadata_elem:
                report_data['metadata'][child.tag] = child.text

        # Сводка по аэропортам
        airports = self.root.findall('.//hub') + self.root.findall('.//regional_airports/airport') +
self.root.findall(
            './/local_airports/airport')
        report_data['airports_summary'] = {
            'total': len(airports),
            'by_type': {
                'hubs': len(self.root.findall('.//hub')),
                'regional': len(self.root.findall('.//regional_airports/airport')),
                'local': len(self.root.findall('.//local_airports/airport'))
            }
        }

        # Сводка по самолетам
        aircraft = self.root.findall('.//aircraft')
        report_data['aircraft_summary'] = {
            'total': len(aircraft),
            'by_type': {}
        }

        for ac in aircraft:
            ac_type = ac.get('type', 'unknown')
            if ac_type not in report_data['aircraft_summary']['by_type']:
                report_data['aircraft_summary']['by_type'][ac_type] = 0
            report_data['aircraft_summary']['by_type'][ac_type] += 1

        with open(output_file, 'w', encoding='utf-8') as f:
            json.dump(report_data, f, indent=2, ensure_ascii=False)

        print(f"JSON отчет создан: {output_file}")

    def generate_yaml_config(self, output_file):
        """Генерация YAML конфигурации для мониторинга"""
        config_data = {
            'monitoring_system': {
                'check_intervals': {
                    'fuel_levels': '15m',
                    'maintenance_schedule': '1h',
                    'security_checks': '30m'
                },
```

```
            'alert_thresholds': {
                'low_fuel': 20,
                'maintenance_due': 7,
                'high_utilization': 90
            }
        },
        'airports_to_monitor': []
    }

    # Добавление аэропортов для мониторинга
    for airport in self.root.findall('.//hub') + self.root.findall('.//regional_airports/airport'):
        airport_data = {
            'id': airport.get('id'),
            'name': airport.find('name').text if airport.find('name') is not None else '',
            'monitoring_points': []
        }

        # Топливные мониторинговые точки
        fuel_elems = airport.findall('.//fuel_capacity') + airport.findall('.//current_fuel')
        if fuel_elems:
            airport_data['monitoring_points'].append('fuel_levels')

        # Мониторинг технического обслуживания
        maintenance_elems = airport.findall('.//maintenance_zones') +
airport.findall('.//maintenance_facilities')
        if maintenance_elems:
            airport_data['monitoring_points'].append('maintenance')

        config_data['airports_to_monitor'].append(airport_data)

    with open(output_file, 'w', encoding='utf-8') as f:
        yaml.dump(config_data, f, default_flow_style=False, allow_unicode=True)

    print(f"YAML конфигурация создана: {output_file}")

def export_to_csv(self, output_file):
    """Экспорт данных в CSV"""
    with open(output_file, 'w', newline='', encoding='utf-8') as csvfile:
        fieldnames = ['id', 'type', 'name', 'location', 'capacity', 'current_passengers', 'utilization_rate']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()

        for airport in self.root.findall('.//hub') + self.root.findall(
                './/regional_airports/airport') + self.root.findall('.//local_airports/airport'):
            capacity_elem = airport.find('terminal_capacity')
            passengers_elem = airport.find('current_passengers')
            name_elem = airport.find('name')
            location_elem = airport.find('location')

            capacity = int(capacity_elem.text) if capacity_elem is not None and capacity_elem.text else 0
            passengers = int(passengers_elem.text) if passengers_elem is not None and passengers_elem.text
else 0
            utilization = (passengers / capacity * 100) if capacity > 0 else 0

            writer.writerow({
                'id': airport.get('id'),
```

```
                'type': airport.tag,
                'name': name_elem.text if name_elem is not None else '',
                'location': location_elem.text if location_elem is not None else '',
                'capacity': capacity,
                'current_passengers': passengers,
                'utilization_rate': f"{utilization:.1f}%"
            })

        print(f"CSV экспорт создан: {output_file}")

    def create_xsd_schema(self, output_file):
        """Создание XSD схемы для валидации"""
        xsd_schema = '''<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="aviation_system">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="metadata">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="created" type="xs:dateTime"/>
              <xs:element name="last_updated" type="xs:dateTime"/>
              <xs:element name="total_airports" type="xs:integer"/>
              <xs:element name="total_aircraft" type="xs:integer"/>
              <xs:element name="security_levels" type="xs:integer"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="airport_hierarchy">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="hubs">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="hub" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="name" type="xs:string"/>
                          <xs:element name="location" type="xs:string"/>
                          <xs:element name="runways" type="xs:integer"/>
                          <xs:element name="terminal_capacity" type="xs:integer"/>
                          <xs:element name="current_passengers" type="xs:integer"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                        <xs:attribute name="region" type="xs:string" use="required"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:decimal" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>'''
```

```
    with open(output_file, 'w', encoding='utf-8') as f:
        f.write(xsd_schema)

    print(f"XSD схема создана: {output_file}")


# ============================================================================
# КЛАСС ДЛЯ ОБРАБОТКИ ОШИБОК И БЕЗОПАСНОСТИ (ЭТАП 5)
# ============================================================================

class AviationSecurity:
    def __init__(self, xml_file):
        self.xml_file = xml_file

    def validate_against_xsd(self, xsd_file):
        """Валидация XML против XSD схемы"""
        try:
            from lxml import etree

            # Загрузка XSD схемы
            with open(xsd_file, 'rb') as f:
                xsd_doc = etree.parse(f)
                xsd = etree.XMLSchema(xsd_doc)

            # Загрузка и валидация XML
            with open(self.xml_file, 'rb') as f:
                xml_doc = etree.parse(f)

            is_valid = xsd.validate(xml_doc)
            validation_errors = []

            if not is_valid:
                for error in xsd.error_log:
                    validation_errors.append(f"Line {error.line}: {error.message}")

            return {
                'is_valid': is_valid,
                'errors': validation_errors
            }

        except ImportError:
            return {'is_valid': False, 'errors': ['Модуль lxml не установлен']}
        except Exception as e:
            return {'is_valid': False, 'errors': [f'Ошибка валидации: {str(e)}']}

    def check_xml_injection(self):
        """Проверка на XML инъекции"""
        injection_patterns = [
            '<!ENTITY',
            '<!DOCTYPE',
            'SYSTEM',
            'PUBLIC',
            '%',
            '&'
        ]
```

```
        try:
            with open(self.xml_file, 'r', encoding='utf-8') as f:
                content = f.read()

            issues = []
            for pattern in injection_patterns:
                if pattern in content.upper():
                    issues.append(f"Обнаружен потенциально опасный паттерн: {pattern}")

            return {
                'has_injections': len(issues) > 0,
                'issues': issues
            }

        except Exception as e:
            return {'has_injections': True, 'issues': [f'Ошибка чтения файла: {str(e)}']}

    def check_reference_integrity(self, analyzer):
        """Проверка целостности ссылок между элементами"""
        integrity_issues = []
        root = analyzer.root

        # Проверка ссылок на аэропорты
        all_airport_ids = set()
        for airport in root.findall('.//hub') + root.findall('.//regional_airports/airport') + root.findall(
                './/local_airports/airport'):
            all_airport_ids.add(airport.get('id'))

        # Проверка родительских ссылок
        for regional in root.findall('.//regional_airports/airport'):
            parent = regional.get('parent_hub')
            if parent and parent not in all_airport_ids:
                integrity_issues.append(f"Неверная ссылка: региональный аэропорт {regional.get('id')} ->
хаб {parent}")

        for local in root.findall('.//local_airports/airport'):
            parent = local.get('parent_airport')
            if parent and parent not in all_airport_ids:
                integrity_issues.append(f"Неверная ссылка: локальный аэропорт {local.get('id')} ->
аэропорт {parent}")

        # Проверка баз самолетов
        for aircraft in root.findall('.//aircraft'):
            base = aircraft.get('base_airport')
            if base and base not in all_airport_ids:
                integrity_issues.append(f"Неверная ссылка: самолет {aircraft.get('id')} -> база {base}")

        return integrity_issues

    def handle_parsing_errors(self):
        """Обработка ошибок парсинга с сохранением контекста"""
        error_context = {
            'file': self.xml_file,
            'timestamp': datetime.now().isoformat(),
            'errors': [],
            'recovery_attempted': False
        }
```

```
    try:
        # Попытка парсинга с восстановлением
        parser = ET.XMLParser(recover=True)
        tree = ET.parse(self.xml_file, parser=parser)

        if parser.error_log:
            for error in parser.error_log:
                error_context['errors'].append({
                    'line': error.line,
                    'column': error.column,
                    'message': error.message
                })
            error_context['recovery_attempted'] = True

        return {
            'success': True,
            'context': error_context,
            'tree': tree
        }

    except Exception as e:
        error_context['errors'].append({'message': f'Критическая ошибка: {str(e)}'})
        return {
            'success': False,
            'context': error_context,
            'tree': None
        }


# ============================================================================
# ОСНОВНАЯ ФУНКЦИЯ
# ============================================================================

def main():
    """Основная функция для выполнения всех этапов"""

    xml_file = "data.xml"

    # Проверка существования файла
    if not os.path.exists(xml_file):
        print(f"❌ Файл {xml_file} не найден!")
        return

    print("=" * 60)
    print("🚀 СИСТЕМА АНАЛИЗА АВИАЦИОННОЙ ИНФРАСТРУКТУРЫ")
    print("=" * 60)

    # ЭТАП 2: Чтение и анализ XML
    print("\n📊 ЭТАП 2: Чтение и анализ XML файла")
    print("-" * 40)

    analyzer = AviationAnalyzer(xml_file)

    # SAX парсинг
    sax_handler = analyzer.parse_with_sax()
    print(f"✅ SAX парсинг: найдено {len(sax_handler.airports)} аэропортов, {len(sax_handler.aircraft)}
```

```
самолетов")

    # Анализ иерархии
    hierarchy = analyzer.analyze_hierarchy()
    print(
        f"✅ Иерархия аэропортов: {len(hierarchy['hubs'])} хабов, {len(hierarchy['regional_airports'])}
региональных, {len(hierarchy['local_airports'])} локальных")

    # Метрики эффективности
    metrics = analyzer.calculate_efficiency_metrics()
    print(
        f"✅ Метрики эффективности: загрузка {metrics['utilization_rate']:.1f}%, топливная
эффективность {metrics['fuel_efficiency']:.1f}%")

    # Карта зависимостей
    dependency_map = analyzer.build_dependency_map()
    print(f"✅ Карта зависимостей: {len(dependency_map)} элементов")

    # ЭТАП 3: Фильтрация и валидация
    print("\n🔍 ЭТАП 3: Фильтрация и валидация")
    print("-" * 40)

    filter_system = AviationFilter(analyzer)

    # Группировка
    grouped_data = filter_system.group_by_type_and_region()
    print(f"✅ Группировка по типам и регионам завершена")

    # Комплексная фильтрация
    filtered_airports = filter_system.filter_by_complex_criteria()
    print(f"✅ Комплексная фильтрация: найдено {len(filtered_airports)} подходящих аэропортов")

    # Проблемы с обслуживанием
    maintenance_issues = filter_system.detect_maintenance_issues()
    if maintenance_issues:
        print(f"⚠️  Обнаружено проблем с обслуживанием: {len(maintenance_issues)}")
        for issue in maintenance_issues[:3]:  # Показать первые 3
            print(f"   - {issue['aircraft_id']}: {issue['issue']} ({issue['days_until']} дней)")
    else:
        print("✅ Проблем с обслуживанием не обнаружено")

    # Проверка целостности
    integrity_issues = filter_system.validate_supply_chain_integrity()
    if integrity_issues:
        print(f"⚠️  Проблемы целостности: {len(integrity_issues)}")
        for issue in integrity_issues[:3]:
            print(f"   - {issue}")
    else:
        print("✅ Целостность цепочек поставок подтверждена")

    # ЭТАП 4: Трансформация и генерация отчетов
    print("\n📈 ЭТАП 4: Трансформация и генерация отчетов")
    print("-" * 40)

    transformer = AviationTransformer(analyzer)
```

```
    # Создание отчетов в разных форматах
    os.makedirs('reports', exist_ok=True)

    transformer.generate_json_report('reports/aviation_report.json')
    transformer.generate_yaml_config('reports/monitoring_config.yaml')
    transformer.export_to_csv('reports/airports_data.csv')
    transformer.create_xsd_schema('reports/aviation_schema.xsd')

    print("✅ Все отчеты и конфигурации созданы")

    # ЭТАП 5: Обработка ошибок и безопасность
    print("\n🛡️ ЭТАП 5: Обработка ошибок и безопасность")
    print("-" * 40)

    security = AviationSecurity(xml_file)

    # Проверка XML инъекций
    injection_check = security.check_xml_injection()
    if injection_check['has_injections']:
        print("❌ Обнаружены потенциальные XML инъекции:")
        for issue in injection_check['issues']:
            print(f"   - {issue}")
    else:
        print("✅ XML инъекции не обнаружены")

    # Проверка целостности ссылок
    reference_issues = security.check_reference_integrity(analyzer)
    if reference_issues:
        print(f"⚠️ Проблемы целостности ссылок: {len(reference_issues)}")
        for issue in reference_issues[:3]:
            print(f"   - {issue}")
    else:
        print("✅ Целостность ссылок подтверждена")

    # Обработка ошибок парсинга
    error_handling = security.handle_parsing_errors()
    if error_handling['success']:
        if error_handling['context']['errors']:
            print(f"⚠️ Обнаружены ошибки парсинга (восстановлено):
{len(error_handling['context']['errors'])}")
        else:
            print("✅ Парсинг завершен без ошибок")
    else:
        print("❌ Критические ошибки парсинга")

    print("\n" + "=" * 60)
    print("✅ ВСЕ ЭТАПЫ ВЫПОЛНЕНЫ УСПЕШНО!")
    print("=" * 60)


if __name__ == "__main__":
    main()
```

Скриншоты выполнения программы: