

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МО и ПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Базовые средства формирования электронных
сообщений защищенного электронного
документооборота

Студент

Зиновьев Леонид Дмитриевич

(подпись, дата)

(ФИО полностью)

Руководитель

Гурьянов Л.В.

(подпись, дата)

(фамилия, инициалы)

Нормоконтролёр

Такташкин Д.В.

(подпись, дата)

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026 г. № 12)

Заведующий кафедрой

Козлов А.Ю.

(подпись)

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

Попова Н.А.

(подпись)

(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Зиновьев Леонид Дмитриевич гр. 22ВП2 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Базовые средства формирования электронных сообщений защищенного электронного документооборота

Тема утверждена приказом ПГУ № _____ от "_____" _____ 20____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Гурьянов Л. В.

4. Задание на работу (назначение разработки, исходные данные и т.п.) _____

1) Назначение: программное средство для работы с электронными сообщениями защищенного электронного документооборота.

2) Среда выполнения: Windows 10, Linux.

Языковые средства разработки: C#, JSON, UML.

Инструментальные средства: Visual Studio 2022, Visual Studio Code, Visual Paradigm.

Технологии разработки: ООП, принципы SOLID, монолитная архитектура.

3) Основные функции приложения:

а) инициализация системы;

б) создание электронного сообщения;

в) отправка заранее подготовленного сообщения;

г) получение электронного сообщения;

д) настройка параметров работы приложения;

е) настройка периодической отправки или создания сообщений;

ж) настройка значений опций команд по умолчанию.

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;

2) Проектирование;

3) Реализация;

4) Контроль качества программного обеспечения;

5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 103 листа, 21 рисунок, 7 таблиц, 1 листинг, 22 использованных источников, 6 приложений.

ЭЛЕКТРОННОЕ СООБЩЕНИЕ, ЗАЩИЩЁННЫЙ ЭЛЕКТРОННЫЙ ДОКУМЕНТООБОРОТ, ЭЛЕКТРОННАЯ ПОДПИСЬ, SOAP, MIME, UML, CLI, C#

Объектом разработки являются программные средства для работы с электронными сообщениями защищённого электронного документооборота.

Целью работы является разработка программного средства, обеспечивающего формирование, отправку, получение и разбор электронных сообщений, а также интеграцию с внешними средствами обработки сообщений и формирования электронной подписи.

Разработка программных средств осуществлялась на языке программирования C#. Для реализации программных средств использовалась среда разработки Visual Studio 2022. Конфигурационные параметры приложения хранятся в файле формата JSON.

Программные средства предназначены для работы на устройствах под управлением ОС Windows 10 и ОС Linux Debian 11

					ПГУ 09.03.04 – 08ВП222.11 ПЗ						
Изм.	Лист	№ докум.	Подп.	Дата	Базовые средства формирования электронных сообщений защищенного электронного документооборота.	Лит.		Лист		Листов	
Разраб.		Зиновьев Л.Д.						4		103	
Пров.		Гурьянов Л.В.									
Реценз.											
Н. контр.		Такташкин Д.В.									
Утверд.		Козлов А.Ю.									
						Группа 22ВП2					

Оглавление

Введение	7
1 Анализ предметной области и постановка задачи на разработку.....	9
1.1 Анализ предметной области	9
1.1.1 Понятие защищённого электронного документооборота	9
1.1.2 Электронные сообщения	9
1.1.3 Требования к структуре, содержанию и защите электронных сообщений.....	10
1.1.4 Проблемы программной обработки электронных сообщений	11
1.2 Анализ существующих подходов к формированию и обработке электронных сообщений.....	12
1.2.1 Форматы представления электронных сообщений	12
1.2.1.1 SOAP	12
1.2.1.2 MIME.....	13
1.2.2 Подходы к обеспечению защиты электронных сообщений	14
1.2.2.1 Электронная подпись	14
1.2.2.2 Дополнительные механизмы защиты данных.....	16
1.3 Постановка задачи на разработку программного средства.....	17
1.4 Функционально-стоимостный анализ.....	19
2 Анализ требований к разрабатываемым программным средствам	24
2.1 Анализ функциональных требований	24
2.2 Анализ нефункциональных требований	32
3 Проектирование программного средства	34
3.1 Высокоуровневое проектирование программного средства	34
3.2 Проектирование главного ядра системы.....	37
3.3 Проектирование модуля формирования сообщений	40
3.4 Проектирование модуля ведения записей.....	42
4 Реализация системы	45
4.1 Реализация ядра системы	45
4.2 Реализация модуля формирования сообщений	48

4.3 Реализация модуля ведения записей	51
5 Контроль качества программного обеспечения	54
5.1 Функциональное тестирование	55
5.2 Модульное тестирование	58
5.3 Интеграционное тестирование	61
5.4 Оценка качества кода	67
5.4.1 Метрики Холстеда	67
5.4.2 Метрики Чепина	71
Заключение	73
Список использованных источников.....	75
Приложение А (справочное) Глоссарий.....	78
Приложение Б (обязательное) Исходный код приложения	81
Приложение В (рекомендуемое) Диаграммы классов.....	88
Приложение Г (рекомендуемое) Результаты функционального тестирования ..	92
Приложение Д (рекомендуемое) Листинг конфигурации системы интеграционного тестирования	98
Приложение Е (рекомендуемое) Акт о внедрении	102

Введение

Современные информационные системы широко применяют электронный документооборот, обмениваясь документами, служебной информацией и сопутствующими данными в электронной форме. По мере развития таких систем растёт и объём передаваемой информации, и число участников взаимодействия, а вместе с этим повышаются требования к надёжности и безопасности электронного обмена.

Ключевым элементом такого взаимодействия выступает электронное сообщение: именно в форме сообщений документы, служебная информация и сопутствующие данные передаются между участниками обмена. В зависимости от назначения и условий использования сообщения могут отличаться по структуре, составу, содержанию и формату представления данных, поэтому их корректное формирование и обработка приобретают особое значение. От характеристик сообщения напрямую зависит, удастся ли его передать, правильно интерпретировать, в дальнейшем проверить и использовать в рамках документооборота.

Разрабатываемое программное средство как раз и предназначено для работы с электронными сообщениями в системе защищённого электронного документооборота. Оно должно корректно формировать сообщения с учётом требований к надёжности и безопасности данных. Его основной сценарий заключается в следующем: приложение получает документ или иные исходные данные из внешнего источника, обрабатывает их и на этой основе формирует электронное сообщение, готовое к последующей передаче. Не менее важно, что решение должно быть пригодно для применения в разных условиях, обеспечивать удобство работы пользователя и поддерживать интеграцию с внешними средствами обработки сообщений и формирования электронной подписи.

Актуальность работы связана с потребностью в инструменте, который упрощает и упорядочивает работу с электронными сообщениями в защищённом документообороте. Такое решение повышает надёжность обмена данными,

снижает вероятность ошибок при обработке сообщений и поддерживает использование внешних средств формирования электронной подписи.

В рамках выпускной квалификационной работы необходимо решить следующие задачи:

- провести анализ предметной области защищённого электронного документооборота;
- определить требования к разрабатываемому программному средству;
- спроектировать архитектуру приложения и интерфейсы взаимодействия его модулей;
- реализовать средства формирования, передачи, получения и проверки электронных сообщений;
- реализовать способы интеграции с внешними средствами формирования электронной подписи и обработки сообщений;
- провести тестирование разработанного программного средства.

Цель выпускной квалификационной работы состоит в разработке базовых средств для защищённого электронного документооборота, обеспечивающих формирование, передачу, получение и проверку электронных сообщений.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области

1.1.1 Понятие защищённого электронного документооборота

В современных информационных системах широко используется электронный документооборот, с помощью которого документы создают, передают, обрабатывают и хранят в электронной форме. Такой подход сокращает время обмена данными и упрощает работу с документами, открывая для них целый ряд новых возможностей [1].

Работа в цифровой среде требует не только организовать обращение с данными, но и обеспечить их защиту [2]. Особое значение здесь приобретают целостность сведений и достоверность их содержания, а также возможность подтвердить источник информации [3]. Чтобы решить эти задачи, применяют средства защиты информации и электронную подпись [4].

Электронная форма взаимодействия заметно отличается от привычного бумажного документооборота и предъявляет особые требования к способам обработки, хранения и передачи данных. Материалы должны быть пригодны для использования в информационных системах, результаты работы с ними должны оставаться достоверными, и при этом важно сохранять возможность их последующей проверки, интерпретации и контроля [3].

С учётом сказанного защищённый электронный документооборот можно понимать как организацию работы с документами в цифровой форме, при которой одновременно обеспечиваются защита данных, надёжность их обработки и подтверждение происхождения передаваемых сведений.

1.1.2 Электронные сообщения

В системах электронного документооборота обмен данными ведётся через электронные сообщения: именно с их помощью передают документы, служебные сведения и прочие данные, нужные для взаимодействия пользователей и

информационных систем. Поэтому электронное сообщение разумно считать основной единицей обмена в цифровой среде.

Электронное сообщение представляет собой не просто набор передаваемых данных: у него есть определённая структура, формат и содержание. Именно эти свойства обеспечивают его корректное представление в системе и дальнейшую программную обработку.

В разрабатываемом программном средстве электронные сообщения формируются в определённом формате, соответствующем RFC 3029, что позволяет однозначно задавать их структуру и правила представления [5].

В структуру сообщения, помимо основного содержимого, могут входить и служебные элементы, нужные для организации обмена. Благодаря этому сообщение можно рассматривать как самостоятельный объект обработки, что особенно важно, когда его должна правильно интерпретировать другая система без участия пользователя.

Следовательно, в системах защищенного электронного документооборота электронные сообщения играют ключевую роль, поскольку именно они обеспечивают передачу данных между участниками взаимодействия и выступают основой последующей обработки информации.

1.1.3 Требования к структуре, содержанию и защите электронных сообщений

Для использования в системах электронного документооборота сообщение должно иметь определенные структуру, состав и содержание. Эти характеристики должны быть заданы таким образом, чтобы сообщение можно было корректно сформировать, передать, принять и обработать в другой системе. Отсутствие единых правил представления данных затрудняет обмен информацией и может привести к ошибкам при ее интерпретации.

Содержимое сообщения должно быть организовано так, чтобы в нем можно было выделить как основные передаваемые данные, так и служебные элементы, необходимые для обработки. Это особенно важно в межсистемном

взаимодействии, где сообщение должно корректно интерпретироваться как пользователем, так и программными средствами.

Отдельного внимания заслуживают требования к защите передаваемой информации [2]. При работе с электронными сообщениями данные необходимо ограждать от несанкционированного воздействия, а кроме того, стоит предусмотреть применение электронной подписи: она позволяет подтвердить происхождение информации и повысить доверие к результатам электронного взаимодействия [4].

Выходит, что электронные сообщения должны быть не только правильно структурированы и представлены: не менее важно обеспечить надёжность их обработки и защиту самой информации.

1.1.4 Проблемы программной обработки электронных сообщений

При программной обработке электронных сообщений, используемых в защищённом документообороте, возникает ряд сложностей. Прежде всего, сообщения могут различаться по структуре и способу представления информации [3], поэтому программное средство должно быть готово работать с ними, учитывая все их особенности.

Кроме того, при разработке подобных средств нельзя упускать из виду удобство их практического применения. Особенно это касается решений, рассчитанных на автоматизированную или командную обработку сообщений: здесь многое зависит от гибкости настройки параметров, возможности повторно выполнять операции и поддержки разных сценариев работы.

Следующий вопрос касается дополнительной информации, которую необходимо получить при обработке сообщения, включая её формирование, передачу и разбор сообщения. Любое искажение этих процессов может привести к невозможности использования полученного сообщения в дальнейшем из-за искажения данных и логики обмена данными.

Таким образом, разработка программного средства для работы с электронными сообщениями требует учета структуры сообщений, надежности

их обработки, интеграции с внешними средствами и удобства практического применения.

1.2 Анализ существующих подходов к формированию и обработке электронных сообщений

1.2.1 Форматы представления электронных сообщений

Формат представления электронного сообщения определяет способ организации данных, правила их структурирования и интерпретации при передаче между информационными системами. От выбранного формата зависят состав сообщения, способ включения служебных данных, возможность автоматизированной обработки и совместимость между различными программными средствами. В системах защищенного электронного документооборота формат сообщения должен обеспечивать не только передачу содержимого, но и возможность его дальнейшей обработки в прикладных сценариях.

Наибольший интерес представляют SOAP и MIME. SOAP используется для представления структурированного содержимого сообщения и задания его логики на уровне межсистемного взаимодействия. MIME, в свою очередь, применяется для представления составного сообщения, включающего несколько частей различного типа, в том числе основное содержимое, вложения и дополнительные данные. Такой порядок позволяет сначала рассмотреть формат основного содержимого, а затем перейти к способу его упаковки в составное сообщение.

1.2.1.1 SOAP

SOAP представляет собой основанный на XML формат обмена структурированными сообщениями между приложениями в распределенной среде. Он задает каркас сообщения и правила его обработки, не определяя прикладную семантику передаваемых данных. Благодаря этому SOAP может

использоваться как универсальный способ представления сообщений в межсистемном взаимодействии.

Сообщение SOAP строится вокруг элемента *Envelope*, который задает границы сообщения. В его составе могут присутствовать *Header* и *Body* [6]. Заголовок используется для передачи служебной информации, связанной с обработкой сообщения, а тело содержит основное прикладное содержимое. Такое разделение позволяет отделить полезные данные от служебных сведений, необходимых для маршрутизации, контроля обработки и других вспомогательных задач.

Для SOAP используется *media type application/soap+xml*, который применяется для обозначения SOAP-сообщений, сериализованных в XML [7]. Это позволяет однозначно идентифицировать формат сообщения при передаче и обработке. В рамках распределенного взаимодействия SOAP также поддерживает формализованный подход к описанию обработки ошибок и организации обмена сообщениями, что делает его удобным для интеграции между различными программными системами.

С практической точки зрения SOAP представляет интерес как формат структурированного представления основного содержимого электронного сообщения. Его использование позволяет формализовать состав сообщения и обеспечить единообразную обработку данных при взаимодействии с внешними системами.

1.2.1.2 MIME

MIME представляет собой механизм расширения формата электронных сообщений, предназначенный для передачи составного содержимого в рамках одного сообщения. Его использование позволяет включать в сообщение несколько частей, отличающихся по типу и назначению, например основное содержимое, вложения и служебные данные.

В основе MIME лежит система заголовков, описывающих характеристики каждой части сообщения. С помощью заголовка *Content-Type* определяется тип

передаваемых данных, а Content-Transfer-Encoding используется для задания способа их представления при передаче [8]. Это позволяет однозначно интерпретировать содержимое отдельных частей сообщения на принимающей стороне.

Наибольшее значение MIME имеет при работе с сообщениями типа multipart [9]. Такой формат позволяет объединять несколько самостоятельных частей в рамках единого контейнера, при этом каждая часть сохраняет собственное описание и может обрабатываться отдельно. Благодаря этому MIME подходит для передачи вложений, бинарных данных и иных дополнительных элементов, которые неудобно включать непосредственно в основное содержимое сообщения.

Таким образом, MIME представляет интерес как формат упаковки составного электронного сообщения, обеспечивающий объединение нескольких частей различного типа в рамках единой структуры.

1.2.2 Подходы к обеспечению защиты электронных сообщений

1.2.2.1 Электронная подпись

Электронная подпись является одним из основных средств обеспечения доверия к электронным данным при межсистемном взаимодействии. Ее применение позволяет подтвердить происхождение данных, обеспечить контроль их целостности и использовать механизмы последующей проверки результата электронного взаимодействия. В условиях защищенного электронного документооборота это особенно важно, поскольку передаваемые сообщения должны быть не только доступны для обработки, но и пригодны для проверки с точки зрения их неизменности и связи с подписавшей стороной.

С точки зрения действующего регулирования электронная подпись рассматривается как информация в электронной форме, присоединенная к другим электронным данным или иным образом связанная с ними и используемая для определения лица, подписавшего информацию. При

электронном взаимодействии она используется как средство подтверждения происхождения данных и контроля их целостности.

Для работы с электронными сообщениями важно не только применение электронной подписи как таковой, но и способ ее представления в составе подписываемых данных. В зависимости от формата сообщения и типа подписываемого объекта могут использоваться различные форматы представления подписи, среди которых наибольшее распространение получили XAdES, CAdES и PAdES. Эти форматы ориентированы на разные типы данных и различаются способом включения подписи в подписываемый объект.

Формат XAdES используется для XML-документов и XML-сообщений [10]. Он основан на XML Signature и предназначен для тех случаев, когда подпись должна быть представлена в рамках XML-структуры и обрабатываться вместе с ней как часть единого документа. Такой подход удобен при работе со структурированными сообщениями, в которых содержимое и подпись должны сохраняться в общей XML-форме.

Формат CAdES применяется для представления подписанных данных в рамках CMS-структур и не привязан только к XML-документам [11]. Он используется в более общем случае, когда подпись рассматривается как криптографическая оболочка над данными и должна быть представлена независимо от их прикладного формата. Такой формат удобен для сценариев, в которых требуется универсальное представление подписанного содержимого. Определение CAdES как CMS Advanced Electronic Signatures закреплено в действующей нормативной базе по электронной подписи.

Формат PAdES рассчитан на документы в формате PDF и применяется тогда, когда подпись нужно встроить прямо в PDF-документ [12]. Он типичен для сценариев, где подписывают электронные документы, предназначенные для визуального просмотра и последующего хранения в виде PDF-файлов. В отличие от XAdES и CAdES, этот формат напрямую завязан на специфику PDF как контейнера документа.

В целом электронная подпись остаётся важным механизмом, который обеспечивает доверие к электронным сообщениям и данным в защищённом электронном документообороте. Значимы здесь и общие принципы её применения, и форматы представления, задающие способ включения подписи в XML-сообщения, в универсальные контейнеры подписанных данных и в PDF- документы.

1.2.2.2 Дополнительные механизмы защиты данных

Наряду с электронной подписью при работе с электронными сообщениями применяются дополнительные механизмы защиты данных, обеспечивающие контроль целостности передаваемой информации и повышение надежности электронного взаимодействия. Их использование позволяет выявлять изменения содержимого сообщения и поддерживать последующую проверку его состояния.

Существенное значение при работе с электронными сообщениями имеет вычисление хеш-значений. Хеш-функция позволяет получить компактное представление данных, используемое для контроля их целостности. Изменение содержимого сообщения приводит к изменению соответствующего хеш- значения, что дает возможность обнаружить нарушение целостности передаваемых данных.

Вычисление хеш-значений применяется и в процедурах формирования и проверки электронной подписи. В таких случаях подпись формируется не по всему содержимому сообщения непосредственно, а по его хеш-значению. Такой подход обеспечивает единообразную подготовку данных к подписанию и упрощает обработку сообщений различного объема.

В системах обработки электронных сообщений указанные операции могут выполняться с использованием внешних средств формирования электронной подписи. При этом прикладное программное средство использует их функциональные возможности без включения внутренней реализации соответствующих механизмов в собственный состав.

Таким образом, дополнительные механизмы защиты данных являются важной частью работы с электронными сообщениями. Их применение позволяет контролировать целостность передаваемой информации и использовать средства формирования электронной подписи в рамках защищенного электронного взаимодействия.

1.3 Постановка задачи на разработку программного средства

По результатам проведенного анализа предметной области необходимо разработать программное средство для работы с электронными сообщениями защищенного электронного документооборота, обеспечивающее их формирование, передачу, получение и разбор.

Защищенный электронный документооборот представляет собой процесс информационного взаимодействия, при котором обмен документами и сопутствующими данными осуществляется в электронной форме с соблюдением требований к надежности и безопасности. Важнейшим элементом такого взаимодействия являются электронные сообщения, служащие средством передачи документов, служебной информации и иных данных между участниками обмена.

Электронные сообщения могут различаться по назначению, структуре и формату представления данных, что делает корректность их формирования, получения и разбора важным условием надежной работы системы электронного документооборота.

Разрабатываемое решение должно обеспечивать выполнение основных операций с электронными сообщениями в рамках защищенного электронного документооборота. При этом необходимо учитывать требования к корректности обработки данных, удобству практического использования и возможности интеграции с внешними средствами обработки сообщений и формирования электронной подписи. Кроме того, разрабатываемое решение должно поддерживать работу в терминальной среде, возможность использования в

различных операционных системах и настройку параметров работы без изменения исходного кода.

На основе проведенного анализа предметной области и существующих подходов необходимо решить следующие задачи:

- определить требования к разрабатываемому программному средству;
- спроектировать архитектуру приложения;
- реализовать основные механизмы работы с электронными сообщениями;
- реализовать способы интеграции с внешними средствами обработки сообщений и формирования электронной подписи;
- провести тестирование разработанного решения.

Разрабатываемая система должна обеспечивать:

- формирование электронных сообщений;
- передачу заранее подготовленных сообщений;
- получение входящих сообщений и извлечение содержащихся в них данных;
- возможность циклического формирования и отправки сообщений по заданным параметрам;
- взаимодействие с внешними средствами обработки сообщений и формирования электронной подписи;
- выполнение операций в терминальном режиме;
- кроссплатформенную работу в операционных системах Windows и Linux;
- настройку параметров работы приложения без изменения исходного кода;
- удобство пользовательского взаимодействия, включая поддержку автоматического дополнения команд.

В результате разработки должно быть создано программное средство для работы с электронными сообщениями защищенного электронного документооборота.

Целью разработки является создание программного средства, обеспечивающего формирование, передачу, получение и разбор электронных сообщений в рамках защищенного электронного документооборота.

1.4 Функционально-стоимостный анализ

Функционально – стоимостной анализ представляет собой способ оценки разрабатываемого программного средства с точки зрения выполняемых им функций и затрат, необходимых для его создания [13]. Применительно к выпускной квалификационной работе такой анализ позволяет определить основные этапы разработки, оценить их продолжительность и рассчитать ориентировочную стоимость выполнения проекта.

Для определения стоимости разработки программного средства необходимо оценить стоимость труда разработчика. В качестве исходных данных может быть использована статистика ожидаемых и предлагаемых заработных плат в сфере информационных технологий за январь – март 2026 года [14]. Соответствующая динамика представлена на рисунке 1.



Рисунок 1 – Динамика ожидаемых и предлагаемых заработных плат в сфере информационных технологий за январь – март за 2026 год в Пензенской области

Из данных рисунка 1 следует, что предлагаемые заработные платы по региону в рассматриваемый период составляют 60 000 рублей в январе, 61 935 рублей в феврале и 70 573 рубля в марте. Среднее значение предлагаемой заработной платы за указанный период составляет 64 169 рублей. При средней продолжительности рабочего месяца 20 рабочих дней, программист получает за каждый день работы приблизительно 3208 рублей. Полученное значение используется при расчете стоимости этапов разработки программного средства.

Для наглядного представления процесса разработки программного средства была построена контекстная диаграмма в нотации IDEF0 [15]. Контекстная диаграмма отражает функцию верхнего уровня, ее входы, выходы, механизмы и управляющие воздействия. Контекстная диаграмма процесса разработки программного средства представлена на рисунке 2.

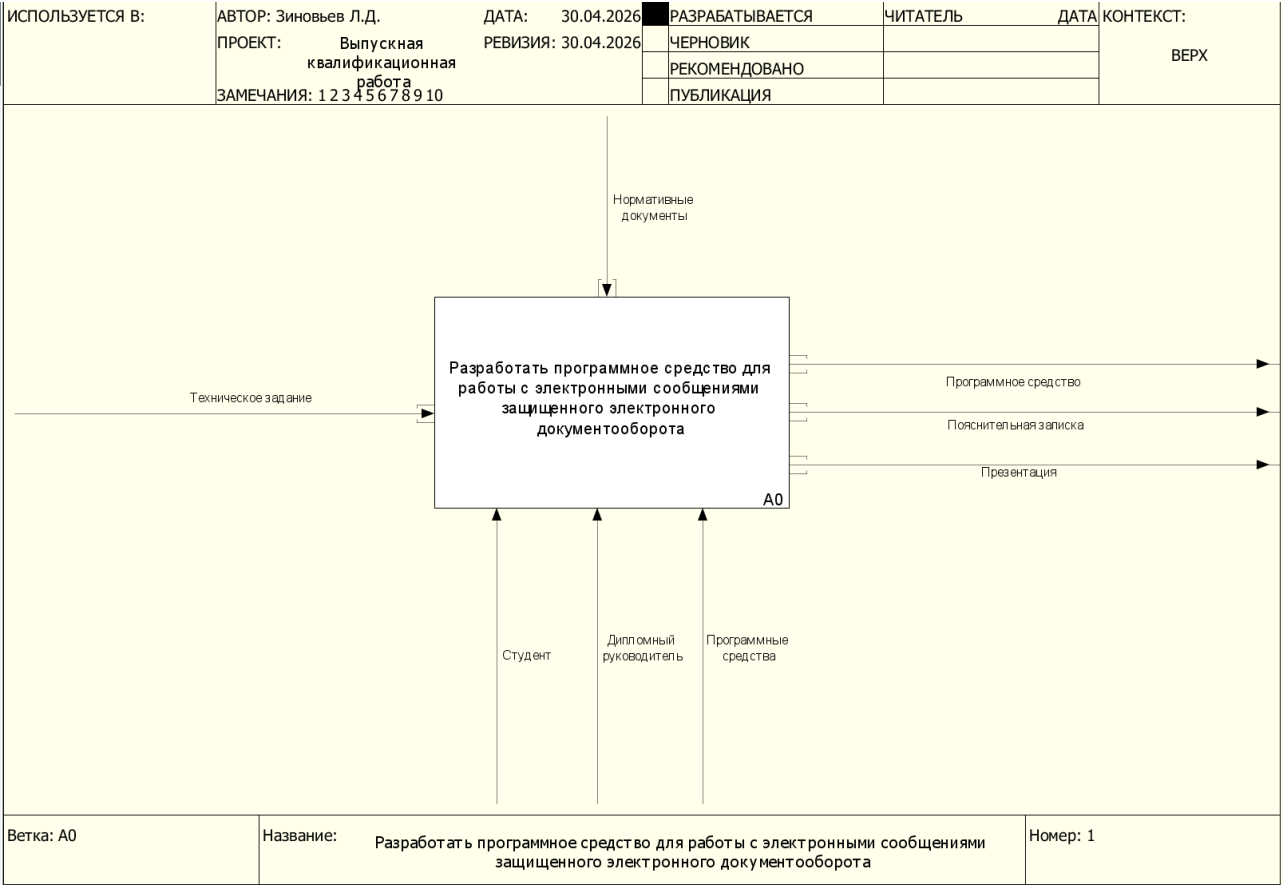


Рисунок 2 – Контекстная диаграмма процесса разработки программного средства

На уровне А-0 функция «Разработать программное средство для работы с электронными сообщениями защищенного электронного документооборота» преобразует входные данные, представленные техническим заданием, под управлением нормативных документов, с использованием знаний студента, консультаций дипломного руководителя и программных средств, в итоговые результаты: программное средство, пояснительную записку и презентацию.

Для более детального анализа процесса разработки была выполнена декомпозиция диаграммы А-0. В результате декомпозиции были выделены основные этапы выполнения выпускной квалификационной работы: анализ предметной области, формирование требований к программному средству, проектирование, разработка, тестирование и оформление пояснительной записки. Декомпозиция функции представлена на рисунке 3.

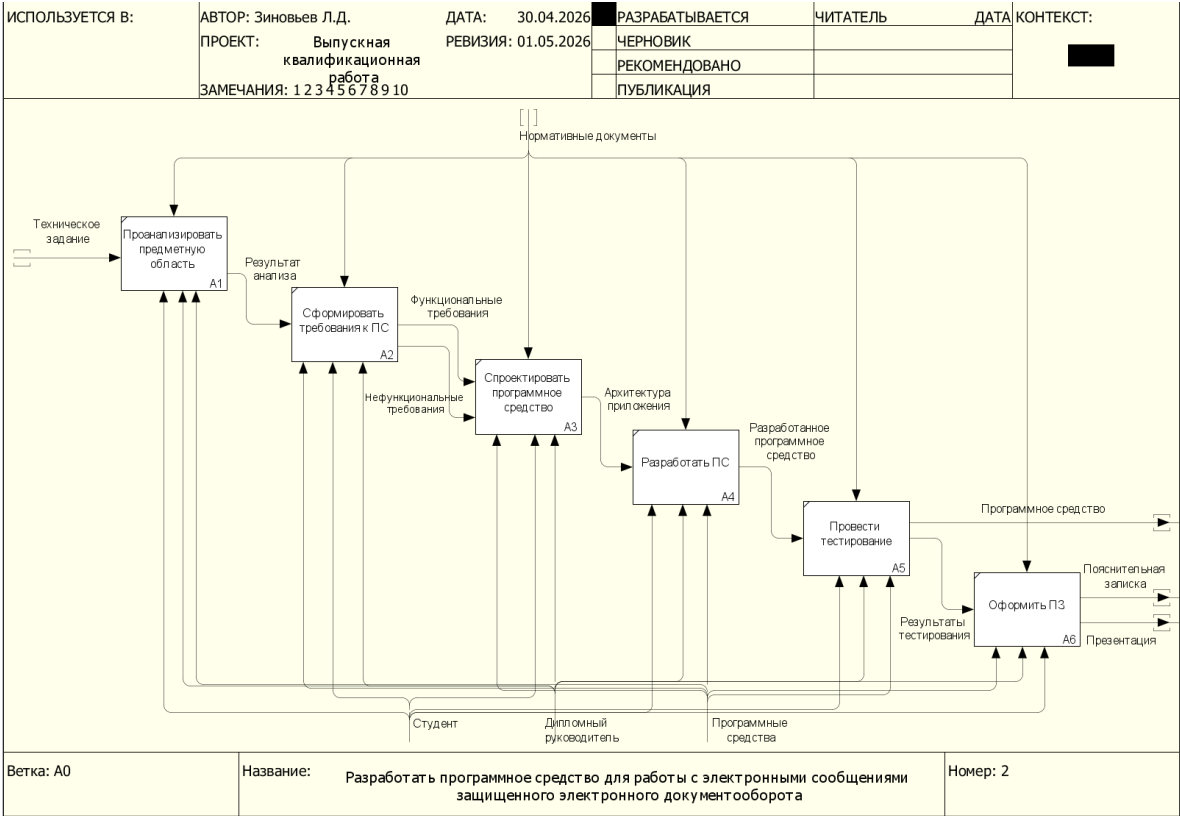


Рисунок 3 – Декомпозиция функции «Разработать программное средство для работы с электронными сообщениями защищенного электронного документооборота»

В результате декомпозиции были выделены шесть подпроцессов: «Проанализировать предметную область», «Сформировать требования к ПС», «Спроектировать ПС», «Разработать ПС», «Провести тестирование» и «Оформить ПЗ». Результаты выполнения предыдущих этапов используются на последующих этапах разработки, что позволяет представить процесс выполнения выпускной квалификационной работы как последовательность взаимосвязанных функций.

Для планирования сроков выполнения проекта был составлен план разработки, отражающий основные этапы работы, их продолжительность и даты начала и завершения. План разработки проекта представлен на рисунке 4.

		Название	Продолжительность	Начало	Окончание
1		☐ Подготовить ВКР	67 дней	05.03.2026, 08:00	05.06.2026, 17:00
2		Проанализировать предметную область	7 дней	05.03.2026, 08:00	13.03.2026, 17:00
3		Проанализировать требования к ПС	5 дней	16.03.2026, 08:00	20.03.2026, 17:00
4		Спроектировать ПС	14 дней	23.03.2026, 08:00	09.04.2026, 17:00
5		Разработать ПС	31 дней	10.04.2026, 08:00	22.05.2026, 17:00
6		Провести тестирование	5 дней	25.05.2026, 08:00	29.05.2026, 17:00
7		Оформить ПЗ	5 дней	01.06.2026, 08:00	05.06.2026, 17:00

Рисунок 4 – План разработки проекта

Для наглядного представления последовательности выполнения этапов разработки и их распределения во времени была построена диаграмма Ганта, представленная на рисунке 5.

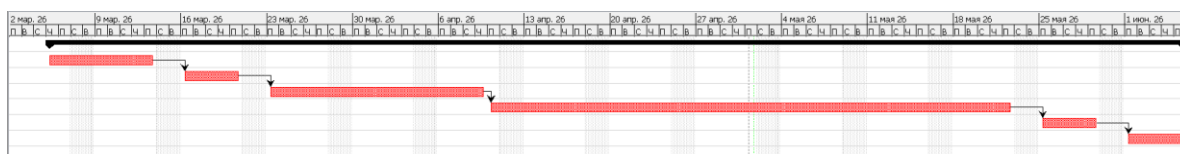


Рисунок 5 – Диаграмма Ганта

Из плана разработки и диаграммы Ганта видно, что наибольшая продолжительность приходится на этап разработки программного средства. Это связано с необходимостью реализации основной функциональности приложения, включая работу с электронными сообщениями, взаимодействие с внешними средствами обработки и защиты данных, а также обеспечение корректной работы приложения в различных сценариях использования.

Для расчета стоимости проекта использовалась продолжительность этапов разработки в рабочих днях. Сводные данные по продолжительности этапов и стоимости их выполнения представлены в таблице 1.

Таблица 1 – Затраты на разработку программного средства

Этап	Продолжительность	Стоимость дня	Стоимость
Проанализировать предметную область	7 дней	3 208 руб./день	22 456 руб.
Сформировать требования к ПС	5 дней	3 208 руб./день	16 040 руб.
Спроектировать ПС	14 дней	3 208 руб./день	44 912 руб.
Разработать ПС	31 день	3 208 руб./день	99 448 руб.
Провести тестирование	5 дней	3 208 руб./день	16 040 руб.
Оформить ПЗ	5 дней	3 208 руб./день	16 040 руб.
Итого	67 дней	3 208 руб./день	214 936 руб.

Из результатов проведенного функционально-стоимостного анализа следует, что суммарная продолжительность разработки программного средства и оформления пояснительной записки составляет 67 рабочих дней, а ориентировочная стоимость выполнения работы равна 214 936 рублей. Наибольшая доля затрат приходится на этап разработки программного средства, что объясняется необходимостью реализации основной функциональности приложения и обеспечения его корректной работы.

Проведённый функционально-стоимостной анализ позволяет оценить трудоёмкость и стоимость разработки программного средства для работы с электронными сообщениями защищённого электронного документооборота, а заодно подтверждает, что выделение основных этапов проекта при планировании выпускной квалификационной работы оправдано.

2 Анализ требований к разрабатываемым программным средствам

2.1 Анализ функциональных требований

Функциональные требования определяют, какие именно функции должно выполнять разрабатываемое программное средство и как система ведёт себя, когда пользователь решает свои задачи. Они фиксируют необходимый функционал системы и описывают, как пользователи будут с ней взаимодействовать. Чёткая формулировка функциональных требований позволяет учесть все цели и задачи работы и довести их до реализации в системе. Благодаря этому снижается вероятность разночтений при их толковании и обеспечивается согласованность проектирования, внедрения и тестирования.

В рамках данной работы создаётся средство для работы с электронными сообщениями. Система должна выполнять основные операции по созданию, отправке, получению и разбору сообщений, а также давать пользователю инструменты настройки параметров работы приложения.

К основным функциональным требованиям, предъявляемым к системе, относятся:

- инициализация системы;
- создание электронного сообщения;
- отправка заранее подготовленного электронного сообщения;
- одновременное создание и отправка электронного сообщения;
- получение электронного сообщения;
- настройка параметров работы приложения;
- получение информации о состоянии системы и подключенных модулей;
- настройка вложения сообщения;
- настройка параметров подписи сообщения;
- настройка подключения при отправке сообщения;
- настройка периодической отправки сообщений;
- возможность задания значений опций команд по умолчанию;

- настройка доступных средств обработки сообщений.

На основе сформулированных функциональных требований была построена диаграмма вариантов использования, представленная на рисунке 6.

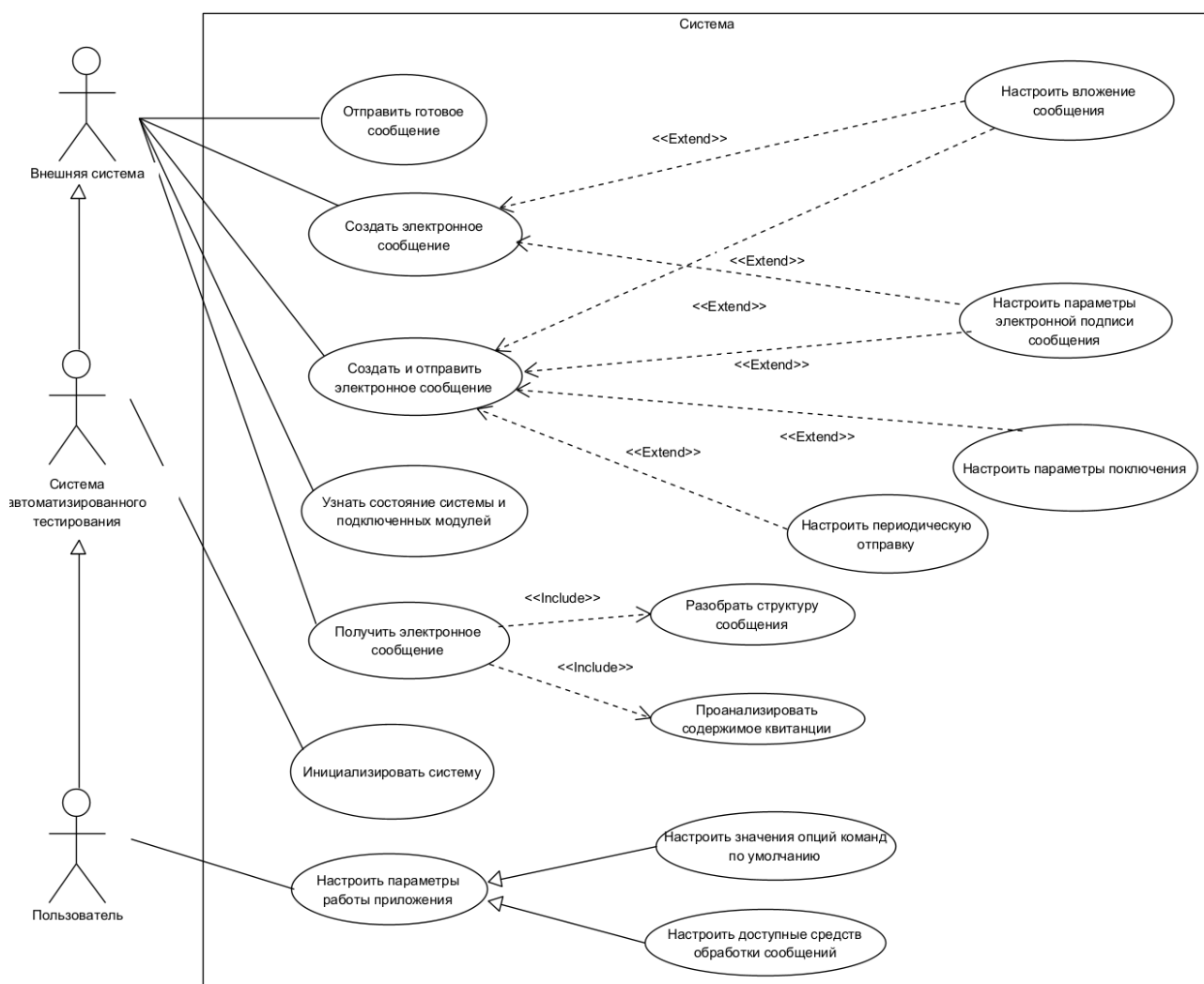


Рисунок 6 – Диаграмма вариантов использования программного средства

Диаграмма вариантов использования предназначена для наглядного представления внешнего взаимодействия с разрабатываемой системой [16]. Она отражает основные сценарии работы приложения, показывает доступные функции и позволяет представить программное средство с точки зрения его внешнего поведения [17].

На диаграмме выделены три внешних актора: пользователь, система автоматизированного тестирования и внешняя система. Пользователь взаимодействует с приложением в ручном режиме через командный интерфейс.

Система автоматизированного тестирования использует приложение для автоматического выполнения проверок и сценариев тестирования. Внешняя система может применять программное средство как вспомогательный инструмент для проверки штатной работы, выполнения автоматизированных сценариев обмена сообщениями и нагрузочного тестирования. При этом основная часть доступного функционала для указанных акторов совпадает, поскольку приложение поддерживает единый набор команд и сценариев работы независимо от способа вызова.

Основу взаимодействия составляют семь базовых сценариев: запуск инициализации, создание электронного сообщения, отправка уже подготовленного сообщения, совмещённое создание с отправкой, получение сообщения, изменение параметров работы приложения и запрос сведений о состоянии системы.

Помимо них предусмотрены вспомогательные сценарии, которые вступают в работу тогда, когда в ходе базовой операции требуется выполнить дополнительную настройку или обработку данных: настроить вложение сообщения, параметры подписи, параметры подключения, режим периодической отправки, а также разобрать структуру сообщения и проанализировать содержимое квитанции.

Наименования всех вариантов использования вместе с краткими пояснениями сведены в таблицу 2.

Таблица 2 – Описание вариантов использования

Актор	Наименование	Описание
Пользователь, система автоматизированного тестирования	Инициализировать систему	Выполняется начальная настройка приложения
Пользователь, система автоматизированного тестирования, внешняя система	Отправить готовое сообщение	Иницируется отправка заранее подготовленного электронного сообщения

Продолжение таблицы 2

Актор	Наименование	Описание
Пользователь, система автоматизированного тестирования, внешняя система	Создать электронное сообщение	Выполняется формирование нового электронного сообщения с учетом заданных параметров
Пользователь, система автоматизированного тестирования, внешняя система	Создать и отправить электронное сообщение	Выполняется формирование электронного сообщения с его последующей отправкой
Пользователь, система автоматизированного тестирования, внешняя система	Получить электронное сообщение	Иницируется получение электронного сообщения для дальнейшего использования
Пользователь	Настроить параметры работы приложения	Изменяются общие параметры функционирования приложения
Пользователь, система автоматизированного тестирования, внешняя система	Узнать состояние системы и подключенных модулей	Извлекается информация о текущем состоянии приложения и доступности подключённых внешних модулей
Пользователь, система автоматизированного тестирования, внешняя система	Настроить вложение сообщения	Задаются параметры вложения, используемого при создании сообщения
Пользователь, система автоматизированного тестирования, внешняя система	Настроить параметры подписи	Задаются параметры подписи, используемой при формировании данных сообщения

Окончание таблицы 2

Актор	Наименование	Описание
Пользователь	Настроить подключения при отправке сообщения	Указываются параметры подключения к доступным средствам обработки сообщений
Пользователь, система автоматизированного тестирования, внешняя система	Настроить периодическую отправку	Задаются параметры циклической отправки сообщений
Пользователь	Настроить значения опций команд по умолчанию	Определяются значения параметров команд, используемых системой по умолчанию
Пользователь	Настроить доступные средства обработки сообщений	Задается перечень доступных средств обработки сообщений, используемых приложением
Пользователь, система автоматизированного тестирования, внешняя система	Разобрать структуру сообщения	Выполняется разбор структуры полученного сообщения и извлечение его основных данных
Пользователь, система автоматизированного тестирования, внешняя система	Проанализировать содержимое квитанции	Выполняется анализ квитанции, полученной в результате разбора сообщения

Сведённая в таблицу информация охватывает как основные, так и вспомогательные сценарии работы внешних акторов с программным средством. По этому перечню видно, какой функционал должна поддерживать система, и именно он ложится в основу дальнейшего проектирования и реализации.

Для части вариантов использования с диаграммы дополнительно расписаны сценарии выполнения. Они подробнее показывают, как акторы

взаимодействуют с приложением и какую последовательность шагов система проходит в рамках ключевых операций.

Сценарий для варианта «Инициализировать систему» приведён на рисунке 7.

Наименование: инициализировать систему
ID: 1
Краткое описание: пользователь инициирует начальную настройку приложения для подготовки его к дальнейшей работе
Основной актер: пользователь
Предусловия: исполняемый файл приложения присутствует и доступен для использования на операционной системе
Основной поток. <ol style="list-style-type: none"> 1. Пользователь инициирует выполнение команды начальной настройки приложения. 2. Приложение определяет среду выполнения и параметры операционной системы. 3. Приложение размещает необходимые служебные скрипты в целевых каталогах системы. 4. Пользователю предоставляется информация об установленных командных файлах и успешном завершении инициализации
Альтернативный поток. <ol style="list-style-type: none"> 1. Пользователь инициирует выполнение команды начальной настройки приложения. 2. В ходе выполнения обнаруживается ошибка размещения служебных файлов или недостаточность прав доступа. 3. Выполнение команды инициализации завершается сообщением об ошибке.
Постусловия: система подготовлена к дальнейшей работе, необходимые служебные файлы и параметры настроены.

Рисунок 7 – Описание варианта использования «Инициализировать систему»

В данном сценарии описывается начальная настройка приложения, выполняемая до начала его дальнейшей эксплуатации. В ходе выполнения сценария программа определяет среду запуска, анализирует параметры операционной системы и размещает необходимые служебные скрипты в целевых каталогах. Такая последовательность действий позволяет подготовить окружение для корректной работы приложения и обеспечить возможность его последующего использования в терминальной среде. По завершении сценария система считается подготовленной к выполнению основных операций.

Как выполняется вариант «Создать электронное сообщение», показано на рисунке 8.

Наименование: создать электронное сообщение
ID: 2.
Краткое описание: пользователь инициирует создание нового электронного сообщения с учетом заданных параметров.
Основной актер: пользователь
Предусловия: приложение инициализировано, параметры работы приложения доступны для использования, внешние модули доступны.
Основной поток. <ol style="list-style-type: none"> 1. Пользователь инициирует выполнение команды создания электронного сообщения. 2. Пользователь указывает необходимые параметры сообщения. 3. При необходимости пользователь задает дополнительные параметры, связанные с вложением, подписью, криптографической защитой и средствами обработки сообщения. 4. Приложение формирует электронное сообщение в соответствии с заданными параметрами. 5. Пользователю предоставляется результат формирования сообщения.
Альтернативный поток. <ol style="list-style-type: none"> 1. Пользователь инициирует выполнение команды создания электронного сообщения. 2. Указанные параметры оказываются неполными, некорректными или противоречивыми. 3. Формирование сообщения не выполняется. 4. Пользователю предоставляется сообщение об ошибке.
Постусловия: электронное сообщение сформировано и доступно для дальнейшего использования, либо пользователю предоставлена информация о причине ошибки.

Рисунок 8 – Описание варианта использования «Создать электронное сообщение»

В данном сценарии описывается формирование нового электронного сообщения на основе заданных параметров. Пользователь указывает основные параметры сообщения, а при необходимости дополнительно задаёт вложение, параметры подписи и параметры подключения. После этого приложение выполняет обработку введенных данных и формирует сообщение в требуемом формате. В результате выполнения сценария создаётся электронное сообщение, готовое к дальнейшему использованию.

Сценарий получения сообщения «Получить электронное сообщение» представлен на рисунке 9.

Наименование: получить электронное сообщение.
ID: 3.
Краткое описание: пользователь инициирует получение электронного сообщения для его дальнейшей обработки.
Основной актор: пользователь.
Предусловия: приложение инициализировано, параметры подключения и обработки сообщения заданы, внешние модули доступны.
Основной поток. 1. Пользователь инициирует выполнение команды получения электронного сообщения. 2. Приложение обращается к источнику получения сообщения с использованием заданных параметров. 3. Электронное сообщение принимается приложением. 4. Приложение выполняет первичную обработку полученных данных. 5. Пользователю предоставляется результат получения сообщения.
Альтернативный поток. 1. Пользователь инициирует выполнение команды получения электронного сообщения. 2. В процессе получения возникает ошибка подключения, обработки или доступа к данным. 3. Получение сообщения не выполняется. 4. Пользователю предоставляется сообщение об ошибке.
Постусловия: электронное сообщение получено и доступно для дальнейшей обработки, либо пользователю предоставлена информация о причине ошибки.

Рисунок 9 – Описание варианта использования «Получить электронное сообщение»

Этот сценарий описывает получение электронного сообщения для его последующей обработки. После запуска соответствующей команды приложение обращается к источнику сообщения с учётом заданных параметров, принимает его, выполняет разбор структуры сообщения и анализирует содержимое полученной квитанции. В результате пользователь либо получает само сообщение, либо видит сведения о возникшей при выполнении ошибке.

Рассмотренные сценарии выполнения позволяют более подробно представить поведение разрабатываемого программного средства при выполнении ключевых действий, с которыми пользователь сталкивается во время работы с приложением. Вместе с диаграммой вариантов использования они дополняют общую картину и помогают точнее сформулировать требования к функциональности системы уже на этапе проектирования.

В итоге проведённый анализ функциональных требований позволил определить состав как основных, так и дополнительных возможностей разрабатываемого программного средства. Полученная диаграмма вариантов использования вместе с описанием ключевых сценариев их выполнения образует основу, на которую опирается дальнейшее проектирование и реализация системы.

2.2 Анализ нефункциональных требований

Нефункциональные требования задают не сам перечень возможностей, а то, как система должна их выполнять. Если функциональные требования отвечают на вопрос «что делает приложение», то нефункциональные описывают его качественные свойства: удобство использования, надёжность, производительность, совместимость и безопасность.

При проектировании такие требования во многом определяют архитектурные решения и то, как будут реализованы отдельные модули, поскольку задают рамки и условия работы системы. Для данного приложения они особенно значимы: оно работает в терминальной среде, должно запускаться в разных операционных системах, взаимодействовать с внешними средствами обработки сообщений и формирования электронной подписи и при этом давать контроль над выполнением отдельных операций.

Ниже перечислены основные нефункциональные требования к системе:

- обеспечение корректной работы приложения в операционных системах Windows и Linux;
- обеспечение выполнения основных операций в терминальном режиме;
- обеспечение возможности настройки параметров работы приложения без изменения исходного кода;
- обеспечение корректной загрузки и использования параметров из конфигурационного файла;

- обеспечение корректного взаимодействия с внешними средствами обработки сообщений и формирования электронной подписи;
- обеспечение устойчивости работы приложения при некорректных входных параметрах и ошибках выполнения команд;
- обеспечение ограничения времени выполнения отдельных операций с помощью настраиваемых таймаутов;
- обеспечение информативного вывода сообщений о результатах выполнения операций и возникающих ошибках;
- обеспечение удобства пользовательского взаимодействия при работе с командным интерфейсом, включая поддержку автоматического дополнения команд и их опций;
- обеспечение возможности выполнения как одиночных, так и циклических операций над сообщениями по заданным параметрам;
- обеспечение расширяемости приложения в части подключения дополнительных средств обработки сообщений;
- обеспечение приемлемого времени выполнения операций формирования, отправки, получения и разбора сообщений;
- обеспечение корректного выполнения процедуры инициализации системы и размещения необходимых служебных файлов.

Перечисленные нефункциональные требования определяют основные качественные характеристики разрабатываемого программного средства и будут использоваться в дальнейшем при проектировании архитектуры приложения, выборе способов реализации отдельных модулей и описании принципов работы системы.

Таким образом, нефункциональные требования определяют основные качественные характеристики разрабатываемого программного средства, включая надежность, совместимость, удобство использования и устойчивость работы приложения. Их учет необходим для выбора архитектурных решений и последующей реализации системы в соответствии с поставленными задачами.

3 Проектирование программного средства

3.1 Высокоуровневое проектирование программного средства

Для разрабатываемого программного средства была выбрана монолитная архитектура с модульной внутренней организацией. Такой подход позволяет реализовать приложение как единое целое, сохранив при этом разделение на логически обособленные компоненты с различными зонами ответственности. Использование данной архитектуры обеспечивает снижение связанности между частями системы, упрощает сопровождение кода и позволяет развивать отдельные части приложения без изменения общей структуры программного средства [18].

Внутренняя организация системы основана на выделении компонентов, отвечающих за обработку командной строки, формирование электронных сообщений, вывод информации и работу с конфигурационными данными. Отдельно выделены внешние системы, используемые для обработки электронных сообщений и выполнения формирования электронной подписи. Такое разделение позволяет отделить прикладную логику приложения от инфраструктурных и интеграционных механизмов, а также обеспечить более гибкое взаимодействие с внешними средствами.

В рамках проектирования были выделены следующие компоненты:

- ядро системы Message.Core;
- компонент формирования электронных сообщений MessageFactory;
- компонент вывода информации OutputManager;
- компонент хранилища конфигурации приложения;
- внешняя система обработки электронных сообщений;
- внешняя система формирования ЭП.

Общая диаграмма компонентов системы представлена на рисунке 10.

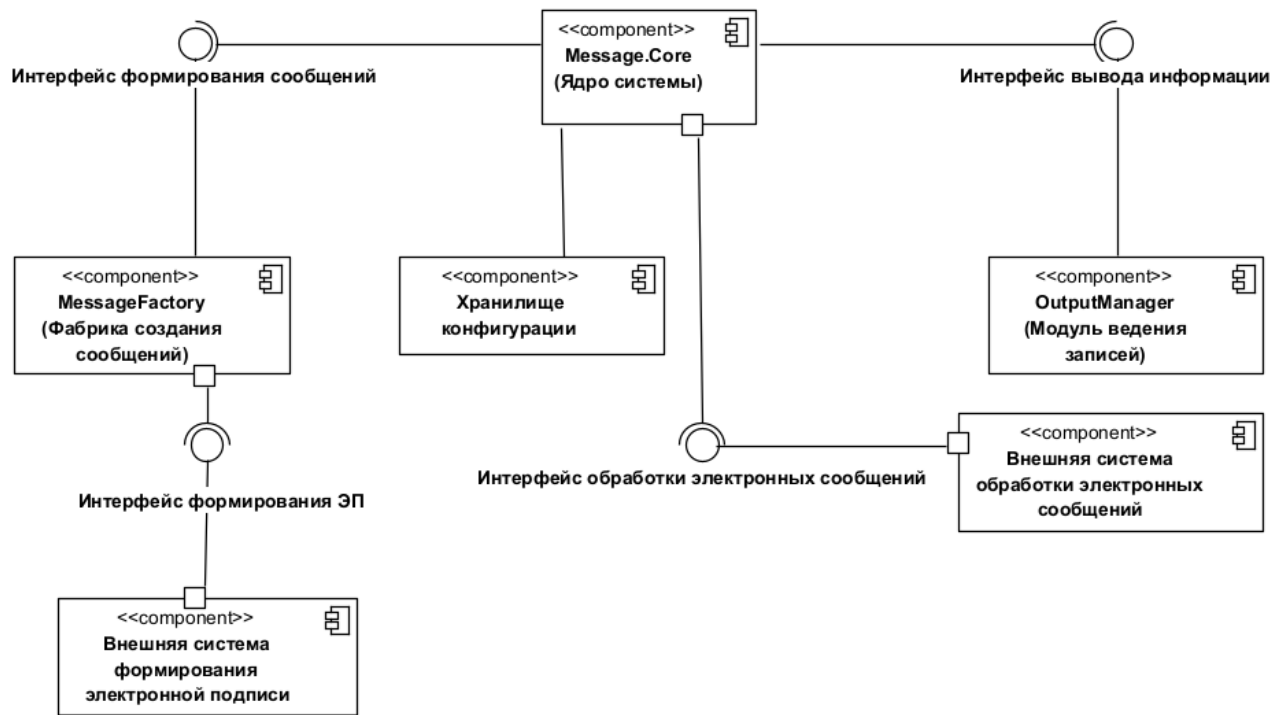


Рисунок 10 – Диаграмма компонентов системы

Ядром системы выступает компонент **Message.Core**. На него возложено выполнение основных операций приложения: он координирует работу внутренних компонентов и одновременно поддерживает связь с внешними системами. В процессе работы **Message.Core** опирается сразу на несколько интерфейсов: через интерфейс формирования сообщений он создаёт электронные сообщения, через интерфейс вывода информации показывает сведения о ходе выполнения операций, а через интерфейс обработки электронных сообщений взаимодействует с внешней системой обработки. Помимо этого, компонент обращается к конфигурационным данным, которые хранятся в хранилище конфигурации.

Компонент **MessageFactory** оформлен как библиотека классов и отвечает за само формирование электронных сообщений. Он получает от ядра уже подготовленные и проверенные параметры и на их основе собирает готовое сообщение. Кроме того, **MessageFactory** взаимодействует с внешней системой формирования ЭП через соответствующий интерфейс, что позволяет

использовать внешние средства подписания без включения их внутренней реализации в состав приложения.

Компонент `OutputManager` отвечает за вывод информации о работе системы. В его задачи входит формирование сообщений разного уровня, управление источниками вывода и настройка того, как именно эта информация отображается. Вынесение вывода в отдельный компонент позволяет отделить прикладную логику от логики представления служебных сообщений, а это, в свою очередь, упрощает сопровождение приложения и делает его поведение более предсказуемым и управляемым.

Хранилище конфигурации используется для хранения и предоставления конфигурационных параметров приложения. В нем содержатся значения параметров команд по умолчанию, а также данные, необходимые для интеграции с внешними средствами обработки электронных сообщений и формирования ЭП. Выделение конфигурационных данных в отдельный источник позволяет изменять параметры работы приложения без модификации исходного кода.

Внешняя система обработки электронных сообщений представляет собой внешний по отношению к приложению компонент, с которым осуществляется взаимодействие через интерфейс обработки электронных сообщений. Аналогичным образом внешняя система формирования электронной подписи подключается через интерфейс и обеспечивает выполнение действий, необходимых при формировании электронных сообщений. Подробные диаграммы классов, отражающие внутреннюю структуру модулей программного средства, приведены в приложении В.

Таким образом, высокоуровневая архитектура программного средства основана на модульной организации монолитного приложения и разделении системы на компоненты, отвечающие за выполнение основных прикладных, сервисных и интеграционных функций. Такой подход позволяет упростить сопровождение программного средства, снизить связанность между его частями и обеспечить возможность дальнейшего развития системы без изменения общей архитектуры.

3.2 Проектирование главного ядра системы

Ядро системы является центральной частью разрабатываемого программного средства и обеспечивает организацию работы приложения в терминальной среде. На уровне проектирования ядро отвечает за разбор входных аргументов, хранение и регистрацию доступных команд, формирование контекста выполнения, настройку внутренних сервисов и передачу управления обработчикам конкретных групп команд.

Структура ядра системы представлена в виде набора классов и интерфейсов, отражающих основные элементы командной инфраструктуры приложения. Диаграмма классов ядра системы приведена на рисунке В.1.

На диаграмме выделяются несколько основных групп классов: классы запуска и первичной настройки приложения, классы разбора аргументов командной строки, классы представления команд и контекста выполнения, классы регистрации команд, а также классы, связывающие зарегистрированные команды с обработчиками соответствующих сценариев.

Точкой входа в приложение служит класс Program. Именно он запускает основной сценарий работы системы, создаёт необходимые объекты главного модуля и передаёт поступившие аргументы командной строки на дальнейшую обработку. Сам по себе Program не реализует прикладную логику выполнения команд, а лишь связывает пользовательский ввод с объектами ядра системы на начальном этапе.

Класс AppOptions хранит параметры конфигурации приложения. В нём собраны настройки, которые используются при подготовке внутренних сервисов и при выборе отдельных режимов работы системы. Вынесение этих параметров в самостоятельный класс отделяет их от логики запуска и в дальнейшем облегчает расширение конфигурационной модели.

Подготовкой внутренних сервисов приложения занимается класс ServiceManager. Опираясь на объект AppOptions, он формирует набор зависимостей, которые нужны остальным компонентам системы для работы. За

счёт этого настройка сервисов оказывается сосредоточена в одном месте и не перемешивается с логикой выполнения пользовательских команд.

Класс `Parser` разбирает аргументы командной строки. Его задача состоит в том, чтобы превратить исходный массив строковых аргументов в структурированный результат разбора. Этот результат представлен классом `ParseArgsResult`, в котором хранятся сведения о выбранной команде, переданных ей параметрах и о том, требуется ли вывод справочной информации.

Класс `Command` описывает отдельную команду приложения. Он содержит имя команды, описание, справочную информацию и перечень поддерживаемых параметров. Кроме того, команда связывается с обработчиком выполнения, благодаря чему ядро системы может единообразно запускать различные команды, не завися от их конкретной внутренней реализации.

Класс `CliApp` является центральным классом ядра системы. Он хранит зарегистрированные команды, предоставляет доступ к ним по имени и выполняет запуск выбранной команды на основе результата разбора аргументов. При запуске команды `CliApp` формирует объект `CliContext`, содержащий необходимые данные выполнения, после чего передает его обработчику команды.

Класс `CliContext` предназначен для хранения контекста выполнения команды. В его состав входят текущая команда, набор переданных параметров и токен отмены операции. Наличие отдельного контекста позволяет передавать обработчикам команд единый объект с необходимыми данными, не усложняя сигнатуры методов большим количеством отдельных параметров.

Интерфейс `ICommandRegister` определяет общий контракт для подключения групп команд к приложению. Он содержит метод регистрации, принимающий объект `CliApp`. Использование интерфейса позволяет унифицировать процесс добавления новых групп команд и отделить механизм регистрации от конкретного набора команд.

Класс `CommandsRegister` выполняет групповую регистрацию команд. Он содержит набор объектов, реализующих интерфейс `ICommandRegister`, и

последовательно вызывает их методы регистрации. Такое решение позволяет централизовать подключение команд к приложению и упростить расширение системы при добавлении новых функциональных групп.

Классы `MessageCommands`, `CryptographyCommands` и `SystemCommands` реализуют интерфейс `ICommandRegister` и отвечают за регистрацию отдельных групп команд. `MessageCommands` подключает команды, связанные с обработкой сообщений, `CryptographyCommands` – команды, относящиеся к функциям формирования ЭП, а `SystemCommands` – системные команды приложения. Каждый из данных классов получает соответствующий обработчик команд через конструктор и использует его при регистрации команд в `CliApp`.

Классы `MessageCommandsResolver`, `CryptographyCommandsResolver` и `SystemCommandsResolver` представляют обработчики соответствующих групп команд. Они получают объект `CliContext` при выполнении команды и используют содержащиеся в нем данные для выбора дальнейшего сценария работы. Такое разделение позволяет оставить ядро системы ответственным за регистрацию, выбор и запуск команд, а прикладную логику выполнения вынести в специализированные обработчики.

Класс `AttachmentManager` используется обработчиком команд сообщений как вспомогательный компонент для работы с вложениями. Он выделен в отдельный класс, чтобы не перегружать обработчик команд сообщениями деталями подготовки файловых данных.

Таким образом, спроектированная структура ядра системы обеспечивает разделение ответственности между основными частями приложения. `Program` отвечает за запуск, `Parser` – за разбор входных аргументов, `CliApp` – за хранение и запуск команд, `CliContext` – за передачу данных выполнения, `CommandRegister` и реализации `ICommandRegister` – за подключение групп команд, а `resolver`-классы – за передачу управления конкретным сценариям обработки. Такая организация делает ядро системы расширяемым и позволяет добавлять новые группы команд без изменения общей схемы запуска приложения.

3.3 Проектирование модуля формирования сообщений

Библиотека создания сообщений предназначена для формирования электронных сообщений на основе подготовленных параметров и данных, полученных из других компонентов системы. Данный модуль реализует прикладную логику подготовки содержимого сообщения и обеспечивает построение итогового представления в зависимости от типа сообщения и используемого формата.

Структура библиотеки создания сообщений была спроектирована в виде набора классов и интерфейсов, отражающих основные сущности, этапы подготовки данных и способы формирования итогового сообщения. Диаграмма классов библиотеки создания сообщений представлена на рисунке В.2.

В состав библиотеки входят классы, отвечающие за хранение входных параметров и промежуточных данных, классы организации процесса подготовки сообщения, а также классы, реализующие формирование итогового сообщения для различных типов и форматов.

Интерфейс IMessageCreator задаёт общий контракт для создания электронного сообщения. В нём объявлен метод Create, который принимает объект MessageData и возвращает уже сформированное представление сообщения. За счёт такого интерфейса удастся выдержать единый подход к созданию сообщений разных типов и форматов.

Класс MessageSettings хранит входные параметры, на основе которых формируется сообщение. В нём собраны сведения о типе взаимодействия, режиме отправки, параметрах подписания, отправителе и получателе, а также данные о вложениях. По сути, MessageSettings выступает исходной моделью параметров, с которой начинается подготовка сообщения.

Класс MessageData описывает уже подготовленные данные сообщения. Здесь хранятся значения, которые напрямую используются при построении итогового представления. Разделение на MessageSettings и MessageData помогает отделить исходные параметры от данных, приведённых к виду, пригодному для дальнейшей работы механизмов создания сообщения.

Класс `ProcessingContext` хранит текущее состояние подготовки сообщения. Он сводит вместе исходные настройки `MessageSettings` и формируемые данные `MessageData`, образуя единый контекст обработки. Благодаря этому необходимую информацию удобно передавать от одного этапа подготовки к другому в рамках одной модели.

Класс `Pipeline` организует сам процесс подготовки сообщения. Он содержит набор шагов обработки и последовательно выполняет те из них, которые применимы к текущему контексту. Вынесение конвейера в отдельный класс отделяет логику подготовки данных от логики построения итогового представления и делает структуру библиотеки более гибкой.

Интерфейс `IProcessingStep` описывает общий контракт для отдельных этапов подготовки сообщения. В нём объявлены методы `CanRun` и `Run`: первый проверяет, применим ли шаг к текущему контексту, а второй выполняет саму обработку. Такой интерфейс задаёт единый способ подключения различных этапов к конвейеру.

Классы `ImmutableDataStep`, `G2GStep`, `B2GStep` и `ConfigurationStep` представляют собой отдельные этапы подготовки данных сообщения. Класс `ImmutableDataStep` используется для формирования общих данных, не зависящих от конкретного сценария. Классы `G2GStep` и `B2GStep` предназначены для подготовки данных, зависящих от типа сообщения. Класс `ConfigurationStep` используется для включения в обработку данных, связанных с конфигурацией и дополнительными параметрами формирования сообщения.

Классы `G2GMessageCreator` и `B2GMessageCreator` являются базовыми классами для двух групп создаваемых сообщений. Они реализуют общие элементы логики формирования сообщений соответствующих типов и задают основу для последующей специализации по формату представления.

Классы `G2GSoapMessageCreator` и `G2GMimeMessageCreator` реализуют создание сообщений типа «government-to-government» в форматах SOAP и MIME соответственно. Аналогичным образом классы `B2GSoapMessageCreator` и `B2GMimeMessageCreator` реализуют создание сообщений типа

«business- to- government» в форматах SOAP и MIME. Такое разделение позволяет учитывать как тип сообщения, так и способ его представления, не смешивая соответствующую логику в одном классе.

В базовых классах создания сообщений предусмотрено и обращение к внешнему сервису подписи. Полученная от него подпись затем используется при построении итогового представления сообщения, благодаря чему подписанные данные включаются в создаваемую структуру.

В целом библиотека создания сообщений объединяет набор классов и интерфейсов, которые отвечают за хранение входных параметров, подготовку данных сообщения и построение итогового представления с учётом его типа и формата. Такая структура распределяет ответственность между основными частями библиотеки, поддерживает разные сценарии формирования сообщений и облегчает дальнейшее развитие модуля.

3.4 Проектирование модуля ведения записей

Модуль ведения записей отвечает за формирование и вывод служебной информации о работе программного средства. С его помощью вывод диагностических, информационных и статусных сообщений управляется централизованно, а способ их представления и направление вывода можно настраивать. Поскольку модуль вынесен в отдельную часть системы, прикладная логика приложения оказывается отделена от логики ведения записей, и сопровождать программное средство становится проще.

Структура модуля ведения записей была спроектирована в виде набора классов и интерфейсов, отражающих общую модель записи, способы ее вывода и параметры конфигурирования. Диаграмма классов модуля ведения записей представлена на рисунке В.3.

В состав модуля входят интерфейсы, задающие общие контракты для трассировки и вывода записей, базовый класс реализации трассировщика, классы конкретных источников вывода, а также вспомогательные сущности, используемые для представления записей и параметров работы модуля.

Интерфейс `ITracer` определяет общий контракт для формирования записей различного уровня. Он содержит методы `Debug`, `Info`, `Warn`, `Error`, а также методы условного вывода `OptionalDebug`, `OptionalInfo`, `OptionalWarn`. Кроме того, интерфейс включает метод `Status`, предназначенный для формирования статусных сообщений о результате выполнения операций. Наличие такого интерфейса позволяет использовать единый способ обращения к модулю ведения записей из других компонентов системы.

Интерфейс `ITracerSink` определяет контракт для компонентов, выполняющих непосредственный вывод записи. Он содержит метод `Write`, принимающий объект `TraceData`. Использование отдельного интерфейса позволяет гибко изменять направление вывода и использовать несколько способов записи информации в рамках одной реализации трассировщика.

Класс `BaseTracer` является основной и единственной реализацией интерфейса `ITracer`. Он обеспечивает формирование записей разных уровней, их фильтрацию в зависимости от параметров конфигурации и последующую передачу в набор подключенных `sinks`. Также данный класс реализует общий механизм записи и методы условного вывода, используемые в тех сценариях, где отображение части сообщений зависит от режима работы приложения.

Класс `TraceData` представляет собой модель отдельной записи. Он содержит уровень записи, текст сообщения и информацию об исключении. Использование отдельного класса для представления записи позволяет унифицировать формат передаваемых данных и обеспечить единый способ их обработки вне зависимости от выбранного источника вывода.

Класс `TracerParameters` хранит параметры конфигурации трассировщика. В нём заданы минимальный уровень вывода, признак работы в частичном режиме отображения, дополнительный параметр `JsonInput` и набор `sinks`, через которые ведётся запись сообщений. Вынесение этих параметров в отдельный класс упрощает настройку трассировщика и даёт возможность управлять его поведением из одного места.

Перечисление `TracerLevelEnum` задаёт уровни сообщений, которые поддерживает модуль ведения записей: `Debug`, `Info`, `Status`, `Warn` и `Error`. За счёт перечисления уровни записей формализованы, и их удобно учитывать при фильтрации и обработке сообщений.

Классы `TerminalTracerSink` и `FileTracerSink` реализуют интерфейс `ITracerSink`: первый выводит записи в терминал, второй записывает данные в файл. Такое разделение позволяет подключать разные источники вывода, не меняя общую логику работы трассировщика.

В итоге модуль ведения записей складывается из набора классов и интерфейсов, которые отвечают за формирование, фильтрацию и вывод служебной информации о работе программного средства. Спроектированная структура позволяет вести работу с диагностическими и статусными сообщениями централизованно, поддерживать разные источники вывода и гибко настраивать параметры ведения записей.

4 Реализация системы

4.1 Реализация ядра системы

Главный модуль системы реализует запуск приложения в терминальной среде, обработку входных параметров, настройку внутренних сервисов и передачу управления обработчикам конкретных сценариев выполнения. В отличие от этапа проектирования, где были определены основные классы и связи между ними, в данном подразделе рассматриваются ключевые механизмы практической реализации ядра приложения.

Отдельное место в реализации главного модуля занимает команда инициализации системы. Ее назначением является подготовка среды выполнения приложения, включая определение используемой операционной системы, выбор соответствующего набора служебных файлов и их размещение в целевых каталогах. В частности, при инициализации обеспечивается установка скриптов автоматического дополнения команд для поддерживаемых оболочек. Последовательность выполнения команды инициализации представлена на рисунке 11.

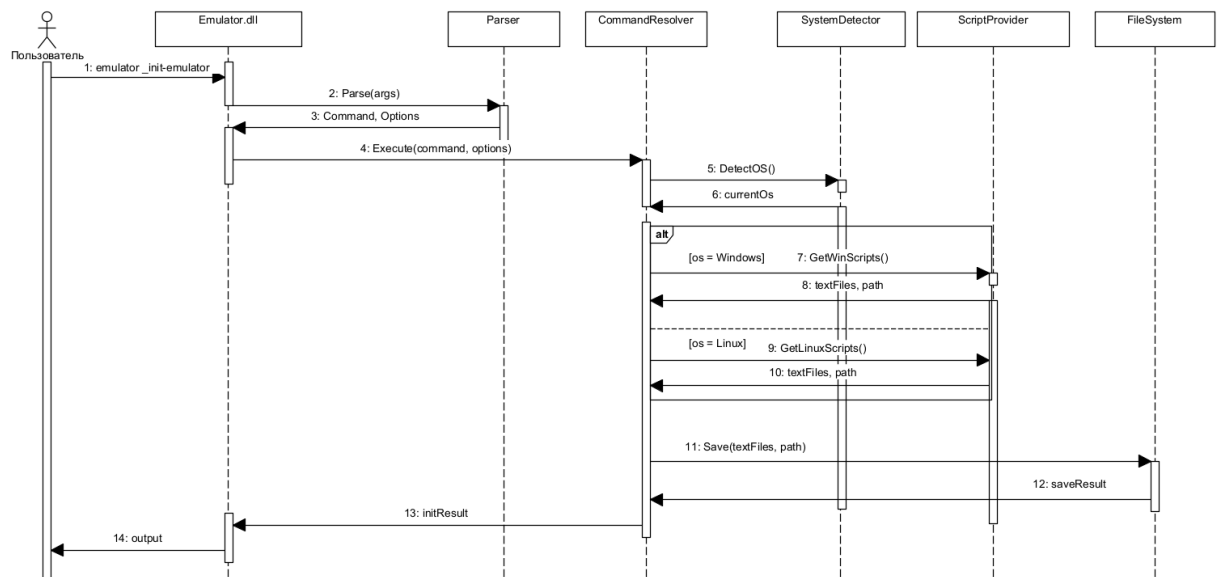


Рисунок 11 – Диаграмма последовательности для команды инициализации

Реализация данного механизма позволяет автоматизировать начальную настройку окружения и упростить последующее использование приложения в

терминальной среде. При этом логика определения операционной системы, выбора набора служебных файлов и их сохранения разделена между отдельными участниками сценария, что делает реализацию более понятной и удобной для сопровождения.

Одной из важнейших задач главного модуля является обработка входных данных, поступающих от пользователя. Для основных несистемных команд в приложении поддерживаются два способа задания параметров: через аргументы командной строки и через JSON-конфигурацию [19]. Независимо от источника данные сначала приводятся к единой промежуточной модели, допускающей наличие незаполненных значений. После этого к ним применяются значения по умолчанию, заданные в конфигурации приложения, затем выполняется этап валидации, а после него этап нормализации. В результате формируется объект *ProcessModel*, содержащий данные безопасных типов, пригодные для дальнейшего выполнения сценария. Общая схема обработки входных параметров представлена на рисунке 12.

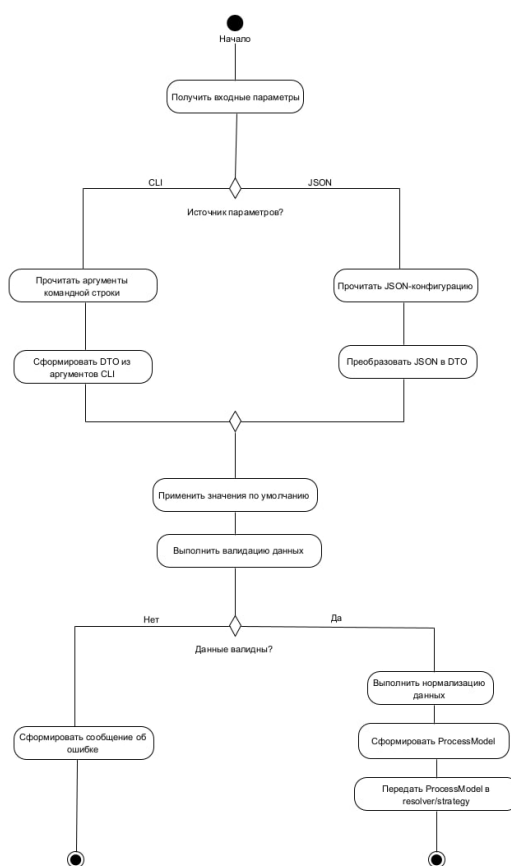


Рисунок 12 – Диаграмма активности обработки входных параметров

Реализованный подход позволяет унифицировать обработку параметров, поступающих из разных источников, и обеспечить единый порядок их подготовки перед выполнением команды. Дополнительным преимуществом такого решения является удобное разрешение случаев взаимно исключающих параметров, поскольку после нормализации данные представляются уже не в виде набора строковых значений, а в виде специализированных структур, отражающих конкретный способ дальнейшей обработки.

Существенной особенностью реализации главного модуля является отказ от использования внешних библиотек командного интерфейса. Для ядра приложения были реализованы собственные механизмы разбора аргументов командной строки, представления команд и их опций, формирования контекста выполнения и вывода справочной информации. Такой подход позволил избежать включения избыточных зависимостей в состав приложения и обеспечить полный контроль над поведением командного интерфейса, логикой обработки параметров и форматом отображаемой справки.

Настройка внутренних сервисов выполняется до запуска основного сценария работы приложения. На начальном этапе из аргументов командной строки извлекаются параметры, необходимые для конфигурирования системы, в том числе уровень ведения записей и настройки трассировки. На их основе формируется модель параметров приложения, которая затем передается в механизм регистрации зависимостей. После подготовки необходимых сервисов, включая компоненты ведения записей и менеджеры взаимодействия с внешними системами обработки сообщений и операций формирования электронной подписи, управление передается основному объекту приложения.

Прикладные сценарии, связанные с обработкой сообщений, выполняются в главном модуле через механизм выбора стратегии. В его основе лежат интерфейс стратегии, базовые реализации исполнителей и фабрика стратегий. Какую именно реализацию задействовать, определяет режим выполнения, переданный в параметрах команды. Выбранная стратегия получает уже подготовленный объект `ProcessModel` и выполняет нужный сценарий обработки.

Сама последовательность выбора стратегии и передачи управления показана на рисунке 13.

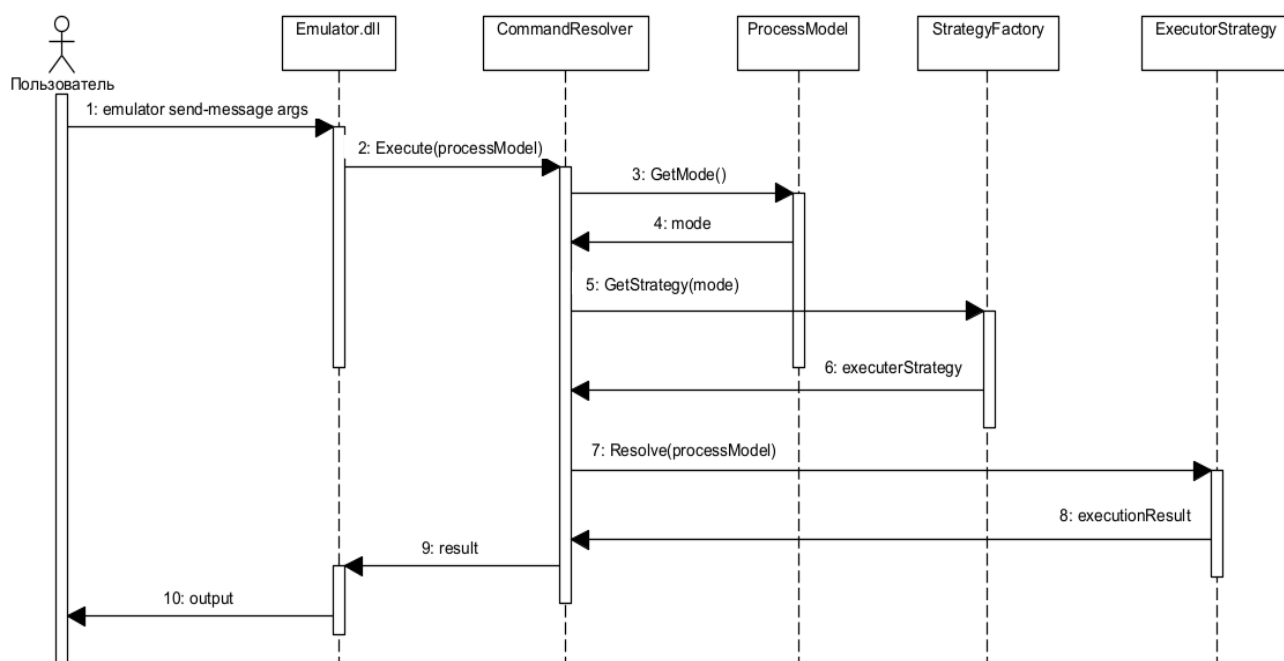


Рисунок 13 – Диаграмма последовательности для выполнения команды отправки сообщения

Фабрика стратегий разводит логику выбора режима работы и логику самого выполнения сценария. Это облегчает расширение приложения при добавлении новых режимов обработки сообщений и снижает связанность между обработчиками команд и конкретными исполнителями.

В итоге в главном модуле собраны основные механизмы, которые отвечают за подготовку среды выполнения, обработку входных параметров, настройку внутренних сервисов и запуск прикладных сценариев. Принятые решения дают единообразное выполнение команд, поддержку нескольких способов задания параметров и возможность расширять приложение, не нарушая целостности его внутренней организации.

4.2 Реализация модуля формирования сообщений

Библиотека создания сообщений реализует прикладную логику подготовки данных и формирования итогового представления электронного сообщения. В

отличие от этапа проектирования, где были определены основные классы и их связи, в данном подразделе рассматриваются особенности практической реализации обработки входных параметров сообщения, конвейерной подготовки данных и построения итогового документа с учетом его типа и формата представления.

Формирование сообщения начинается с обработки объекта MessageSettings, содержащего исходные параметры сценария. На его основе в библиотеке создается контекст обработки, объединяющий входные настройки и формируемые данные сообщения. После этого управление передается объекту MessagePipeline, который организует последовательное выполнение этапов подготовки данных. Общая последовательность формирования электронного сообщения представлена на рисунке 14.

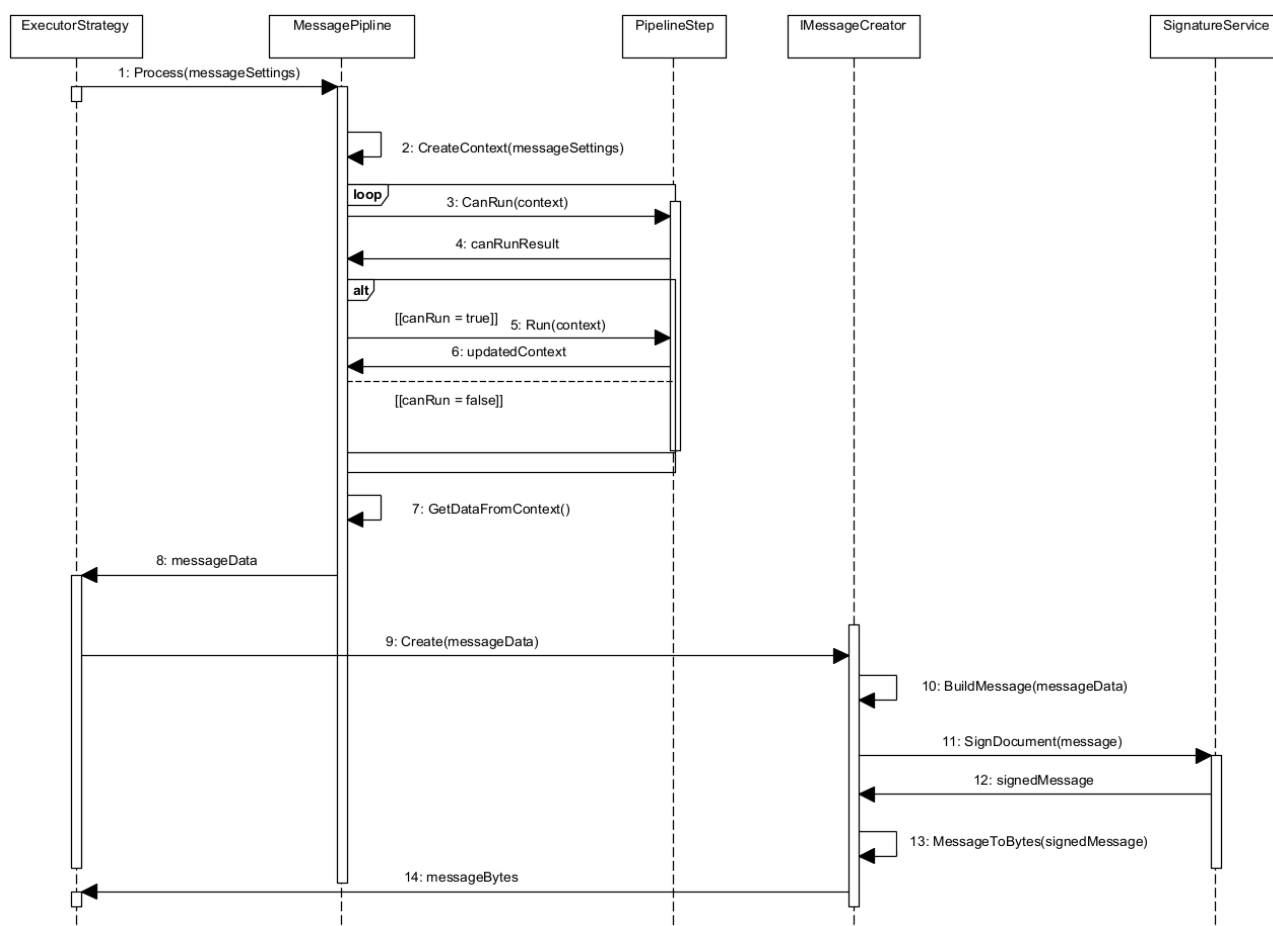


Рисунок 14 – Диаграмма последовательности для выполнения формирования электронного сообщения

Подготовка данных сообщения реализована с использованием конвейерного подхода. После создания контекста обработки объект `MessagePipeline` последовательно обращается к доступным шагам обработки. Каждый шаг перед выполнением проверяет свою применимость к текущему состоянию контекста с помощью метода `CanRun`. Если шаг соответствует текущему сценарию, выполняется его обработка, в противном случае данный этап пропускается. Такой механизм позволяет использовать единый набор шагов для различных сценариев формирования сообщений и при этом выполнять только те преобразования, которые действительно необходимы.

Итогом работы конвейера становится объект `MessageData`, в котором данные уже приведены к виду, пригодному для непосредственного формирования сообщения. Отдельный этап подготовки позволяет развести между собой преобразование и обогащение исходных параметров и логику построения итогового документа. Вдобавок такой порядок упрощает работу с взаимно исключающими параметрами: после нормализации данные представлены специализированными структурами, каждая из которых отражает конкретный способ дальнейшей обработки.

Когда данные готовы, управление переходит к объекту, реализующему интерфейс `IMessageCreator`. На этом шаге на основе объекта `MessageData` строится итоговая структура сообщения. Конкретная реализация создателя выбирается исходя из типа сообщения и формата его представления, но внешний сценарий работы библиотеки от этого не меняется и остаётся единообразным.

Важная особенность реализации заключается в том, что операции подписи вынесены во внешний сервис. Сформировав содержимое сообщения, библиотека обращается к `SignatureService`, получает от него результат подписания и встраивает его в формируемый документ. После этого сообщение переводится в байтовое представление, которое и возвращается вызывающей стороне как итог работы библиотеки.

Реализованная схема формирования сообщения обеспечивает поддержку различных сценариев без дублирования общей логики подготовки данных.

Конвейерная организация обработки, условное выполнение этапов и использование общего интерфейса создателя сообщения позволяют сохранить расширяемость библиотеки при добавлении новых типов сообщений, форматов представления и правил обработки.

Таким образом, реализация библиотеки создания сообщений основана на последовательной подготовке данных, условном выполнении этапов обработки, построении итоговой структуры сообщения и взаимодействии с внешним сервисом подписи. Такой подход обеспечивает поддержку различных сценариев формирования сообщений, сохраняет модульность внутренней организации библиотеки и упрощает ее дальнейшее развитие.

4.3 Реализация модуля ведения записей

Модуль ведения записей предназначен для централизованного формирования и вывода служебной информации о работе программного средства. В отличие от этапа проектирования, где были определены основные классы и интерфейсы данного модуля, в данном подразделе рассматриваются особенности практической реализации механизма записи сообщений, их фильтрации и передачи в различные источники вывода.

Основной точкой взаимодействия с модулем является вызов методов интерфейса трассировки, соответствующих различным уровням сообщений. При обращении к модулю ведения записей формируется объект `TraceData`, содержащий уровень сообщения, его текст и при необходимости сведения об исключении. Такой подход позволяет унифицировать представление служебной информации и использовать единый формат записи независимо от её источника и назначения. Если запись удовлетворяет установленным условиям фильтрации, она передаётся в подключённые источники вывода. После этого выполняется проверка допустимости вывода записи в соответствии с установленным минимальным уровнем трассировки. Общая последовательность обработки записи представлена на рисунке 15.

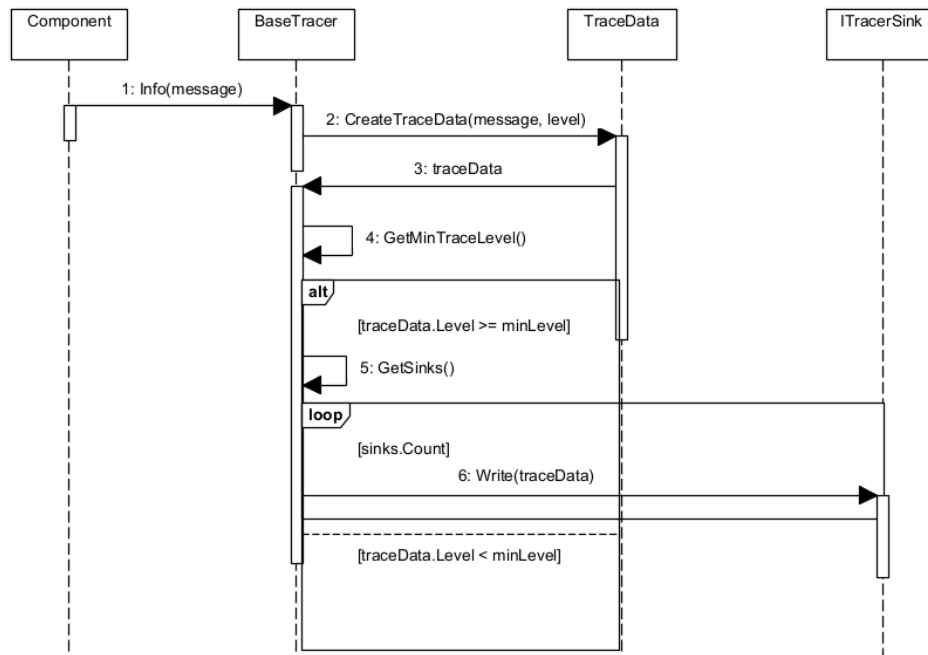


Рисунок 15 – Диаграмма последовательности для выполнения записи процесса выполнения команды

Центральным элементом реализации является класс `BaseTracer`, который принимает сообщения от других компонентов системы, формирует на их основе объект `TraceData`, выполняет проверку уровня записи и организует дальнейший вывод. Если уровень сообщения не ниже установленного минимального значения, запись передается во все подключенные источники вывода. В противном случае вывод сообщения не выполняется.

Использование механизма фильтрации по уровню позволяет управлять объемом выводимой служебной информации без изменения логики работы других компонентов системы. В зависимости от настроек приложения модуль может выводить только наиболее важные сообщения либо поддерживать более подробный режим трассировки, включающий отладочную информацию.

После прохождения проверки запись передается в набор объектов, реализующих интерфейс `ITracerSink`. Такой подход позволяет отделить логику формирования и фильтрации сообщений от логики их непосредственного вывода. В результате один и тот же объект `TraceData` может быть одновременно направлен в несколько источников, например в терминал и в файл.

Реализация вывода через интерфейс источников делает модуль ведения записей гибким и расширяемым. Подключение нового источника вывода не требует изменения общей логики формирования сообщений и выполняется за счет добавления новой реализации интерфейса ITracerSink. Это позволяет использовать единый механизм ведения записей в различных сценариях работы приложения.

Дополнительной особенностью реализации является поддержка нескольких уровней сообщений, включая отладочные, информационные, статусные, предупреждающие и ошибочные записи. Такое разделение упрощает восприятие служебной информации пользователем и позволяет более точно настраивать поведение системы при выводе сообщений.

Настраивается модуль ведения записей через объект параметров, в котором заданы минимальный уровень трассировки, режим частичного вывода и набор подключённых источников записи. За счёт этого поведением модуля можно управлять ещё на этапе конфигурирования приложения, не трогая исходный код.

В итоге работа модуля ведения записей строится на трёх началах: записи формируются централизованно, фильтруются по уровню важности и передаются в набор настраиваемых источников вывода. Такая организация отделяет механизм ведения записей от прикладной логики, оставляет настройку гибкой и позволяет одновременно выводить служебную информацию сразу в несколько сред.

5 Контроль качества программного обеспечения

Тестирование остаётся неотъемлемой частью разработки программного обеспечения. Оно подтверждает, что отдельные части приложения работают правильно, и проверяет, насколько корректно компоненты системы взаимодействуют друг с другом. Регулярное тестирование помогает вовремя находить ошибки, повышать качество продукта и поддерживать его стабильную работу при эксплуатации [20].

В разработке используется сразу несколько видов тестирования, и они различаются целями, методами, а также своими сильными и слабыми сторонами. Какой именно вид применить, зависит от задач проекта, стадии жизненного цикла системы и предъявляемых к продукту требований по качеству. В рамках выпускной квалификационной работы были рассмотрены наиболее распространённые из них и выбраны подходы, подходящие для данного случая.

Функциональное тестирование проверяет, соответствует ли программное обеспечение заданным функциональным требованиям. Опираясь на спецификации и требования к системе, оно показывает, выполняет ли продукт все заявленные функции. Фактически такой вид тестирования воспроизводит реальное использование системы и помогает находить логические ошибки в реализации функциональности. При этом обнаруживать проблемы с производительностью, надёжностью или безопасностью оно не предназначено. Главная его задача состоит в том, чтобы убедиться, что нужные функции выполняются корректно, и выявить логические дефекты в работе программы.

Интеграционное тестирование сосредоточено на том, насколько правильно отдельные компоненты программного обеспечения работают вместе, и проводится уже после модульного тестирования. С его помощью выявляют ошибки, которые возникают именно при совместной работе модулей, и оценивают, как система ведёт себя как единое целое. Сильная сторона такого подхода заключается в способности находить дефекты, связанные с обменом данными и логикой взаимодействия компонентов. В то же время он бывает трудоёмким и иногда требует дополнительных ресурсов и специальной

инфраструктуры. Его цель состоит в том, чтобы обеспечить согласованную работу компонентов и выявить ошибки интеграции.

Модульное тестирование проверяет отдельные компоненты или модули программы и обычно выполняется самими разработчиками ещё на этапе реализации. Оно позволяет ловить ошибки на ранних стадиях и облегчает дальнейшее сопровождение кода, поскольку даёт возможность быстро убедиться в работоспособности отдельных частей системы после очередного изменения. Однако ошибки во взаимодействии между модулями оно не выявляет, а на написание и поддержку тестов уходит дополнительное время. Основная задача модульного тестирования состоит в том, чтобы проверить, что каждый отдельный модуль работает правильно и выполняет свои функции так, как задумано.

5.1 Функциональное тестирование

Для функционального тестирования программного средства был выбран метод чёрного ящика [21]. При таком подходе поведение системы оценивают по входным данным и полученным результатам, не анализируя внутреннюю реализацию её компонентов. Использование данного метода позволяет проверить корректность выполнения основных функций приложения, оценить обработку пользовательских сценариев и установить соответствие фактического поведения системы предъявляемым требованиям. В рамках тестирования были рассмотрены как штатные сценарии работы, так и ошибочные ситуации, возникающие при передаче некорректных, неполных или противоречивых параметров. Код модульных тестов приведён в приложении Б. Проверка охватывала команды, связанные с созданием, отправкой, получением и обработкой электронных сообщений, а также служебные операции инициализации системы и получения информации о состоянии внешних подсистем. Результаты тестирования представлены в приложении Г. На основе выбранного метода была составлена таблица тестов, охватывающая ключевые

функциональные возможности разрабатываемого программного средства. Описание функционального тестирования представлено в таблице 3.

Таблица 3 – Результаты функционального тестирования программного средства

№	Проверяемая команда	Входные данные	Ожидаемый результат	Результат
1	Автоматическое формирование и отправка сообщения	send-message --mode AutoSend --mime	Сообщение должно быть сформировано и передано. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, сообщение было сформировано и передано, пользователю выведена информация о ходе выполнения (рисунок Г.1)
2	Формирование сообщения с подписью XAdES	send-message --mode Create --outputpath /home/msg --sign-interface XAdES	Сообщение должно быть корректно сформировано по заданным параметрам и сохранено по указанному пути. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, сообщение сформировано и сохранено, пользователю выведена информация о ходе выполнения (рисунок Г.2)
3	Формирование сообщения типа B2G с подписью CAdES	send-message --mode Create --outputpath /home/msg --sign-interface CAdES	Команда завершит свою работу с кодом 0.	Команда успешно завершила выполнение с кодом 0 и информацией о ходе выполнения (рисунок Г.2)
4	Формирование сообщения типа B2G с подписью PAdES	send-message --mode Create --outputpath /home/msg --sign-interface PAdES	Сообщение должно быть корректно сформировано и сохранено. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, сообщение сформировано и сохранено, пользователю выведена информация о ходе выполнения (рисунок Г.2)
5	Инициализация системы на Linux	_init-emulator --log-level Debug	Система должна быть корректно инициализирована, служебные файлы должны быть размещены в определённых директориях. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, инициализация системы выполнена, пользователю выведена информация о ходе выполнения (рисунок Г.3)
6	Формирование сообщения с сохранением в файл	send-message --mode Create --outputpath /home/test-msg	Сообщение должно быть сформировано и сохранено по указанному пути. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, сообщение сформировано и сохранено, пользователю выведена информация о ходе выполнения (рисунок Г.4)

Продолжение таблицы 3

№	Проверяемая команда	Входные данные	Ожидаемый результат	Результат
7	Формирование сообщения с настройкой вложения	send-message --mode Create --outputpath /home/test-msg --size 15MB --log-level Debug	Сообщение должно быть сформировано с учетом заданных параметров вложения. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, сообщение сформировано, пользователю выведена информация о ходе выполнения (рисунок Г.5)
8	Отправка ранее подготовленного сообщения	send-message --mode CompleteMessageSend --inputpath /home/test-msg --log-level Debug	Подготовленное сообщение должно быть обработано и отправлено. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, сообщение обработано и отправлено, пользователю выведена информация о ходе выполнения (рисунок Г.6)
9	Проверка состояния средства формирования ЭП	status-crypto --log-level Debug	Должна быть выведена информация о состоянии средств формирования ЭП. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, информация о состоянии подсистемы формирования ЭП выведена пользователю (рисунок Г.7)
10	Проверка состояния системы обработки сообщений	status-queue --log-level Debug	Должна быть выведена информация о состоянии внешней системы обработки сообщений. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, информация о состоянии системы обработки сообщений выведена пользователю (рисунок Г.8)
11	Обработка отсутствия обязательного параметра	send-message --mode CompleteMessageSend	Должно быть выведено сообщение об ошибке. Команда не должна завершаться успешно	Команда корректно обработала ошибочную ситуацию и вывела сообщение об отсутствии обязательного параметра (рисунок Г.9)
12	Обработка недопустимого значения параметра	send-message --mode Auto --size 150MB	Должно быть выведено сообщение об ошибке. Команда не должна завершаться успешно	Команда корректно обработала ошибочную ситуацию и вывела сообщение о недопустимом значении параметра (рисунок Г.10)
13	Получение сообщения	receive-message --segmentName Gateway-Emulator --queue-name OUT.QUEUE --outputpath /home/msg -ll Debug	Сообщение должно быть успешно получено, разобрано и сохранено	Команда успешно завершила выполнение с кодом 0, сообщение получено и сохранено (рисунок Г.11)
14	Циклическая отправка	Send-message --loop	Должно быть отправлено несколько сообщений	Команда успешно сформировала, отправила и вывела пользователю информацию о сообщениях (рисунок Г.12)

Окончание таблицы 3

№	Проверяемая команда	Входные данные	Ожидаемый результат	Результат
15	Инициализация системы Windows	_init-emulator	Система должна быть инициализирована, служебные файлы должны быть размещены. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, инициализация системы выполнена, пользователю выведена информация о ходе выполнения (рисунок Г.13)
16	Проверка состояния системы обработки сообщений на Windows	status-queue --segmentName Gateway-Emulator	Должна быть выведена информация о состоянии внешней системы обработки сообщений. Команда должна завершиться с кодом 0	Команда успешно завершила выполнение с кодом 0, информация о состоянии системы обработки сообщений выведена пользователю (рисунок Г.13)

Функциональное тестирование показало, что программное средство корректно обрабатывает основные пользовательские сценарии: формирование, отправку, получение и обработку электронных сообщений, а также выполняет системные команды и проверяет состояние внешних подсистем. Тесты подтвердили, что и штатные, и ошибочные ситуации обрабатываются правильно, включая отсутствие обязательных параметров и передачу недопустимых значений. Также была проверена корректная работа как в операционной системе Windows, так и Linux. Всё это говорит о том, что фактическое поведение программного средства соответствует установленным функциональным требованиям.

5.2 Модульное тестирование

Чтобы убедиться в корректности работы отдельных частей программного средства, было проведено модульное тестирование. Если функциональное тестирование смотрит на поведение системы в рамках пользовательских сценариев, то модульное проверяет отдельные методы и компоненты приложения по отдельности, в изоляции. Такой подход помогает находить ошибки в логике обработки данных, проверять отдельные алгоритмы и убеждаться, что система ведёт себя правильно при разных входных значениях.

В ходе модульного тестирования проверке подверглись отдельные элементы главного модуля и библиотеки формирования сообщений. Общий результат прогона показан на рисунке 16.

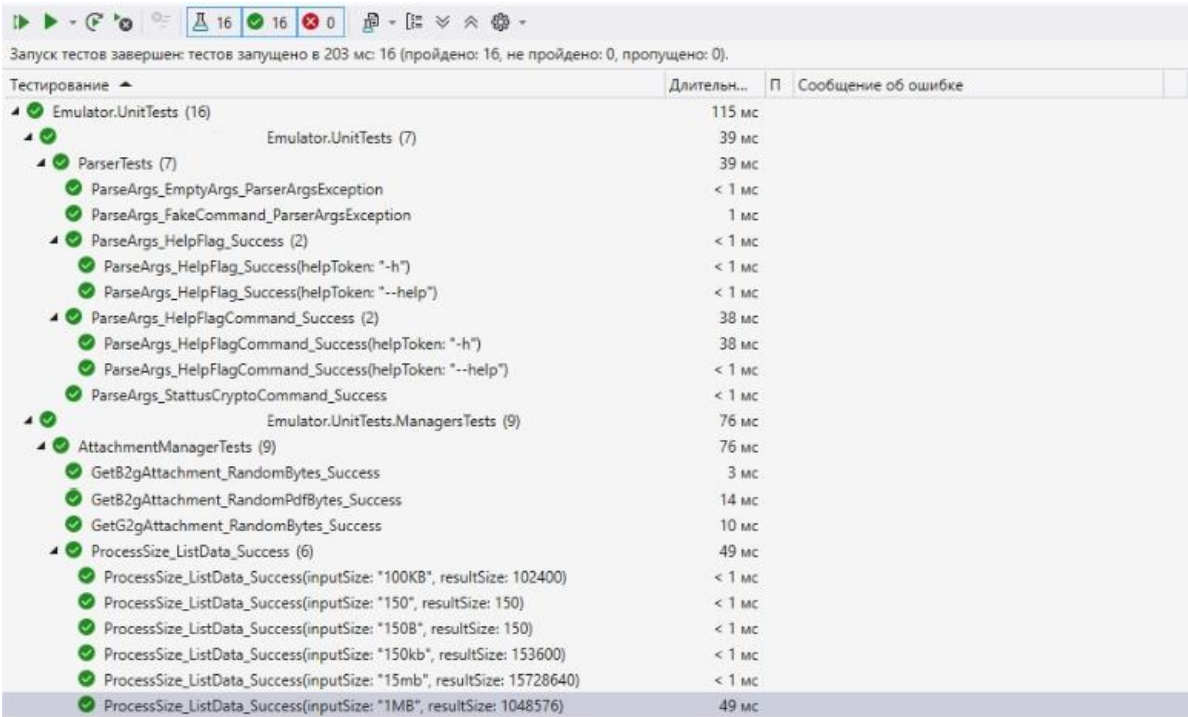


Рисунок 16 – Результат модульного тестирования

Описание каждого модульного теста приведено в таблицах 4 и 5.

Таблица 4 – Результаты модульного тестирования модуля обработки аргументов командной строки

Тест	Описание	Ожидаемый результат	Результат
ParseArgs_EmptyArgs_ParserArgsException	Проверка обработки вызова парсера без переданных аргументов	Должно быть сгенерировано исключение ParserArgsException	Успешно
ParseArgs_FakeCommand_ParserArgsException	Проверка обработки неизвестной команды	Должно быть сгенерировано исключение ParserArgsException	Успешно
ParseArgs_HelpFlag_Success(helpToken: «-h»)	Проверка обработки флага справки без указания команды	Парсер должен корректно определить запрос справки	Успешно
ParseArgs_HelpFlag_Success(helpToken: «--help»)	Проверка обработки полного флага справки без указания команды	Парсер должен корректно определить запрос справки	Успешно

Продолжение таблицы 4

Тест	Описание	Ожидаемый результат	Результат
ParseArgs_HelpFlagCommand_Success(helpToken: «-h»)	Проверка обработки краткого флага справки для конкретной команды	Парсер должен корректно определить команду и признак вывода справки	Успешно
ParseArgs_HelpFlagCommand_Success(helpToken: «--help»)	Проверка обработки полного флага справки для конкретной команды	Парсер должен корректно определить команду и признак вывода справки	Успешно
ParseArgs_StatusCryptoCommand_Success	Проверка корректного разбора команды status-crypto	Команда должна быть успешно распознана и преобразована в результат разбора	Успешно

Данные таблицы 4 показывают, что модуль обработки аргументов командной строки корректно справляется и со штатными сценариями, и с ошибочными случаями: отсутствием аргументов, неизвестными командами, запросом справки.

Таблица 5 – Результаты модульного тестирования библиотеки формирования сообщений

Тест	Описание	Ожидаемый результат	Результат
GetB2gAttachment_RandomBytes_Success	Проверка формирования вложения типа B2G на основе случайных байтов	Должен быть сформирован корректный объект вложения	Успешно
GetB2gAttachment_RandomPdfBytes_Success	Проверка формирования PDF-вложения типа B2G на основе случайных байтов	Должен быть сформирован корректный объект PDF-вложения	Успешно
GetG2gAttachment_RandomBytes_Success	Проверка формирования вложения типа G2G на основе случайных байтов	Должен быть сформирован корректный объект вложения	Успешно
ProcessSize_ListData_Success(inputSize: «100KB», resultSize: 102400)	Проверка преобразования значения размера 100KB	Значение должно быть корректно преобразовано в 102400 байт	Успешно
ProcessSize_ListData_Success(inputSize: «150», resultSize: 150)	Проверка преобразования числового значения без указания единицы измерения	Значение должно быть интерпретировано как 150 байт	Успешно
ProcessSize_ListData_Success(inputSize: «150B», resultSize: 150)	Проверка преобразования значения 150B	Значение должно быть корректно преобразовано в 150 байт	Успешно

Продолжение таблицы 5

Тест	Описание	Ожидаемый результат	Результат
ProcessSize_ListData_Success(inputSize: «100kb», resultSize: 102400	Проверка преобразования значения 100kb	Значение должно быть корректно преобразовано в 102400 байт	Успешно
ProcessSize_ListData_Success(inputSize: «15mb», resultSize: 15728640	Проверка преобразования значения 15mb	Значение должно быть корректно преобразовано в 15728640 байт	Успешно
ProcessSize_ListData_Success(inputSize: «1MB», resultSize: 1048576	Проверка преобразования значения 1MB	Значение должно быть корректно преобразовано в 1048576 байт	Успешно

Данные таблицы 5 подтверждают, что методы библиотеки формирования сообщений, отвечающие за создание вложений и обработку параметров размера, работают корректно. Тесты показывают, что эти методы правильно обрабатывают разные варианты входных данных и дают ожидаемый результат.

В итоге модульное тестирование подтвердило корректную работу отдельных компонентов программного средства. По его результатам можно заключить, что базовые механизмы разбора аргументов командной строки и подготовки данных для формирования электронных сообщений реализованы правильно.

5.3 Интеграционное тестирование

Для проверки корректности совместной работы основных компонентов программного средства было проведено интеграционное тестирование. В отличие от модульного тестирования, ориентированного на проверку отдельных методов и компонентов в изолированном виде, интеграционное тестирование позволяет оценить согласованность выполнения целостных сценариев работы приложения, включая запуск команд, обработку параметров, взаимодействие с внешними подсистемами и получение итогового результата.

Для проведения интеграционного тестирования было разработано собственное вспомогательное приложение, предназначенное для автоматизированного запуска команд тестируемого программного средства и

формирования отчёта о результатах выполнения. Код конфигурации системы представлен в приложении Д.

В рамках интеграционного тестирования были проверены как отдельные команды приложения, так и сценарии прохождения тракта сообщения. Процесс интеграционного тестирования представлен на рисунке 17.

```
12345:~# dotnet /home/tester/IntegrationTestSystem.dll /home/tester/tester-config.jsonc

Система интеграционного тестирования
проверка команд программы через DLL

Конфигурация <configuration>/tester-config.jsonc
Целевая DLL <external>/Emulator.dll

Запуск проверок

OK Статус очередей.
шаг 1/13, код 0, 5033 мс
OK Статус криптографии.
шаг 2/13, код 0, 264 мс
OK Отправить сообщение.
шаг 3/13, код 0, 605 мс
OK Отправить сообщение.
шаг 4/13, код 0, 645 мс
OK Отправить time сообщение.
шаг 5/13, код 0, 5370 мс
OK Очистить очередь.
шаг 6/13, код 0, 5182 мс
OK Создать подписанный документ.
шаг 7/13, код 0, 391 мс
OK Базовый сценарий тракта сообщения. / Очистить очередь.
шаг 8/13, код 0, 460 мс
OK Базовый сценарий тракта сообщения. / Отправить сообщение.
шаг 9/13, код 0, 585 мс
OK Базовый сценарий тракта сообщения. / Получить сообщение.
шаг 10/13, код 0, 519 мс
OK Базовый сценарий тракта time сообщения. / Очистить очередь.
шаг 11/13, код 0, 461 мс
OK Базовый сценарий тракта time сообщения. / Отправить time сообщение.
шаг 12/13, код 0, 5224 мс
OK Базовый сценарий тракта time сообщения. / Получить сообщение.
шаг 13/13, код 0, 5188 мс

ИТОГ ПРОЙДЕНО
всего 13 пройдено 13 ошибки 0 медленно 0
HTML-отчет <external>/integration-report.html
JSON-отчет <configuration>/reports/integration-report.json
12345:~#
```

Рисунок 17 – Процесс интеграционного тестирования

На рисунке 17 показана общая последовательность выполнения интеграционного тестирования. Сначала вспомогательное приложение загружает конфигурацию тестового запуска, содержащую перечень команд и сценариев, после чего последовательно инициирует выполнение проверок для тестируемого программного средства. Для каждой команды фиксируются код завершения,

длительность выполнения, стандартный вывод и сведения о возможных ошибках. На основе полученных результатов формируется итоговый отчёт, позволяющий оценить корректность выполнения отдельных команд и согласованность работы приложения в рамках целостных сценариев.

По результатам тестового запуска было выполнено 13 проверок, из которых все 13 завершились успешно. Ошибок и медленных операций в рамках тестового прогона зафиксировано не было. Сводные результаты интеграционного тестирования представлены на рисунке 18.

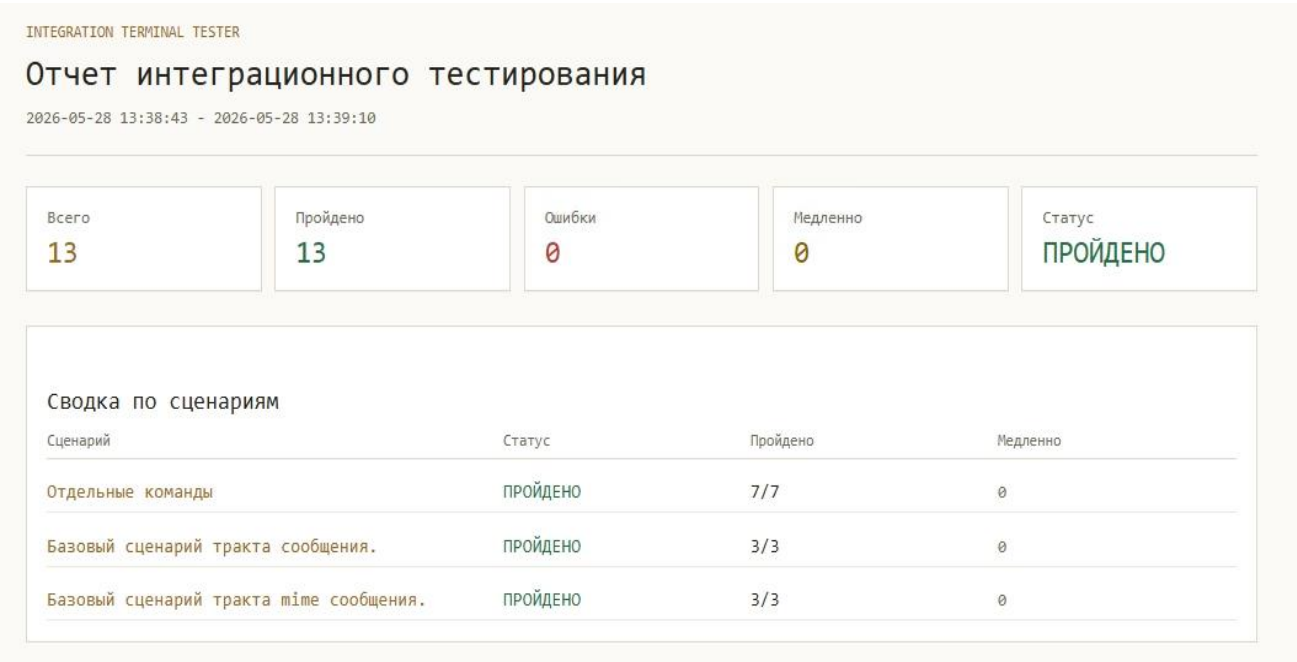


Рисунок 18 – Общий отчёт интеграционного тестирования

На рисунке 18 показана сводная информация по выполненному тестовому запуску, включая общее количество проверок, число успешно пройденных тестов, отсутствие ошибок и медленных операций, а также итоговый статус выполнения.

Детальные результаты проверки отдельных команд представлены на рисунке 19.

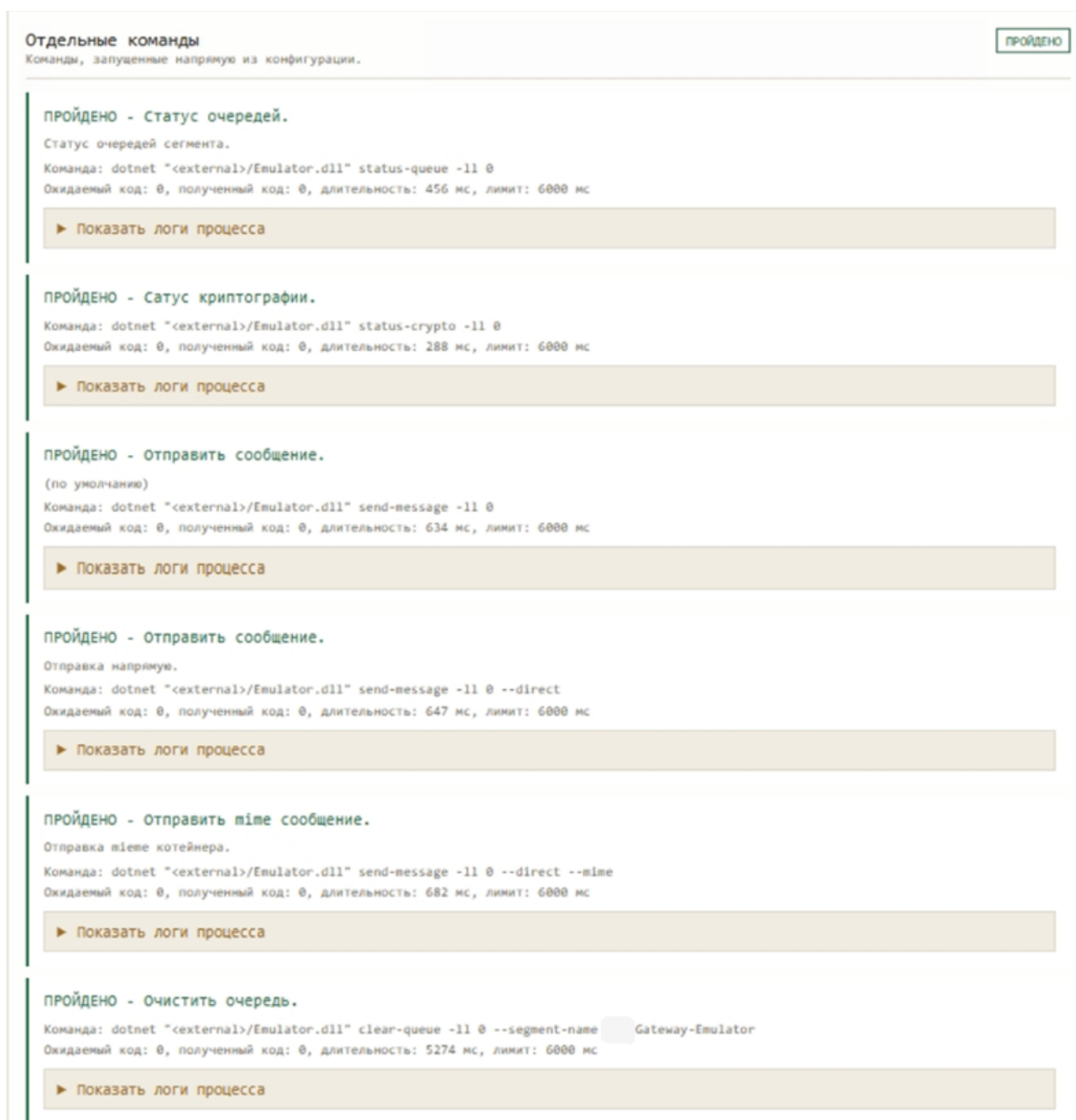


Рисунок 19 – Результаты проверки отдельных команд

На рисунке 19 показаны результаты выполнения отдельных команд программного средства: проверки состояния внешних подсистем, отправки сообщений, очистки очереди и создания подписанного документа. Эти данные подтверждают, что отдельные операции приложения отрабатывают корректно в рамках интеграционного тестирования.

Результаты прохождения базового сценария тракта сообщения показаны на рисунке 20.

Сценарий: Базовый сценарий тракта сообщения.

Последовательность команд, выполненная как единый сценарий.

ПРОЙДЕНО

ПРОЙДЕНО - Очистить очередь.

Команда: dotnet "<external>/Emulator.dll" clear-queue -ll 0 --segment-name Gateway-Emulator
Ожидаемый код: 0, полученный код: 0, длительность: 449 мс, лимит: 6000 мс

▼ Показать логи процесса

STDOUT

```
[13:38:51] [ debug ] Старт операции: QueueClear
[13:38:51] [ debug ] Старт проверки подключения к очереди сегмента: Gateway-Emulator.
[13:38:52] [ status ] Статус очередей сегмента Gateway-Emulator: [Подключены]
[13:38:52] [ debug ] Конец проверки подключения к очереди: Gateway-Emulator.
[13:38:52] [ debug ] Очередь .IN.QUEUE сегмента Gateway-Emulator очищена: 0/0
[13:38:52] [ debug ] Очередь .OUT.QUEUE сегмента Gateway-Emulator очищена: 0/0
[13:38:52] [ debug ] Очередь IN.QUEUE сегмента Gateway-Emulator очищена: 0/0
[13:38:52] [ debug ] Очередь OUT.QUEUE сегмента Gateway-Emulator очищена: 0/0
[13:38:52] [ result ] ok
[ClearQueueResult]
[data] Очередь IN.QUEUE очищена : 0/0
[data] Очередь .OUT.QUEUE очищена : 0/0
[data] Очередь IN.QUEUE очищена : 0/0
[data] Очередь .OUT.QUEUE очищена : 0/0
[13:38:52] [ debug ] Завершение операции: QueueClear
```

STDERR

Вывод отсутствует.

ПРОЙДЕНО - Отправить сообщение.

Отправка напрямую.

Команда: dotnet "<external>/Emulator.dll" send-message -ll 0 --direct
Ожидаемый код: 0, полученный код: 0, длительность: 5259 мс, лимит: 6000 мс

► Показать логи процесса

ПРОЙДЕНО - Получить сообщение.

Режим: только внутри сценариев

Команда: dotnet "<external>/Emulator.dll" receive-message -ll 0 --segmentName Gateway-Emulator --queue-name OUT.QUEUE --outputpath /home/test-msg
Ожидаемый код: 0, полученный код: 0, длительность: 495 мс, лимит: 6000 мс

► Показать логи процесса

Рисунок 20 – Результаты выполнения базового сценария тракта сообщения

Здесь видно, как последовательно выполняются команды базового сценария обработки сообщения: очистка очереди, отправка сообщения и его последующее получение. Представленная последовательность отражает прохождение сообщения через основные этапы обработки и демонстрирует корректное взаимодействие команд в рамках одного сценария. Это подтверждает, что в пределах единого сценария команды работают согласованно.

Результаты прохождения базового сценария тракта MIME-сообщения приведены на рисунке 21.

Сценарий: Базовый сценарий тракта mime сообщения.
Последовательность команд, выполненная как единый сценарий.

ПРОЙДЕНО

ПРОЙДЕНО - Очистить очередь.

Команда: dotnet "<external>/Emulator.dll" clear-queue -ll 0 --segment-name Gateway-Emulator
Ожидаемый код: 0, полученный код: 0, длительность: 472 мс, лимит: 6000 мс

► Показать логи процесса

ПРОЙДЕНО - Отправить mime сообщение.

Отправка mime котейнера.

Команда: dotnet "<external>/Emulator.dll" send-message -ll 0 --direct --mime
Ожидаемый код: 0, полученный код: 0, длительность: 5410 мс, лимит: 6000 мс

► Показать логи процесса

ПРОЙДЕНО - Получить сообщение.

Режим: только внутри сценариев

Команда: dotnet "<external>/Emulator.dll" receive-message -ll 0 --segmentName Gateway-Emulator --queue-name OUT.QUEUE --outputpath /home/test-msg
Ожидаемый код: 0, полученный код: 0, длительность: 5180 мс, лимит: 6000 мс

▼ Показать логи процесса

STDOUT

```
[13:39:05] [ debug ] Для команды задан таймаут: 10000 мс.
[13:39:05] [ debug ] Старт получения сообщения.
[13:39:05] [ debug ] Старт проверки подключения к очереди сегмента: Gateway-Emulator.
[13:39:10] [ status ] Статус очередей сегмента Gateway-Emulator: [Подключены]
[13:39:10] [ debug ] Конец проверки подключения к очереди: Gateway-Emulator.
[13:39:10] [ info ] Количество сообщений в очереди OUT.QUEUE: 1.
[13:39:10] [ debug ] Тип сообщения определен как XorMessage.
[13:39:10] [ debug ] Роль сообщения: Sender
[13:39:10] [ info ] Сообщение успешно получено и сохранено: /home/test-msg.
[13:39:10] [ debug ] Конец получения сообщения.
```

STDERR

Вывод отсутствует.

Рисунок 21 – Результаты выполнения базового сценария тракта MIME-сообщения

На рисунке отражено последовательное выполнение команд, связанных с обработкой MIME-сообщения. Этот сценарий подтверждает, что формирование, отправка и получение MIME-сообщения проходят корректно в рамках интеграционного тестирования.

В итоге интеграционное тестирование подтвердило, что основные компоненты программного средства согласованно работают вместе как при выполнении отдельных команд, так и при прохождении целостных сценариев обработки сообщений. Полученные результаты говорят о согласованности реализованных механизмов и о работоспособности приложения при комплексном выполнении основных операций.

5.4 Оценка качества кода

5.4.1 Метрики Холстеда

Среди подходов к количественной оценке программного кода выделяется метрика Холстеда, основанная на анализе используемых в программе операторов и операндов. Данный подход позволяет оценить объём программы, сложность её реализации и предполагаемые затраты на понимание и сопровождение кода. В отличие от простых количественных метрик, таких как SLOC, метрика Холстеда ориентирована на оценку логической структуры программной реализации.

Для расчёта метрик Холстеда рассматривается функция `ParseArguments` компонента парсинга аргументов командной строки. Функция реализует анализ массива токенов, проверку корректности имени команды, обработку опций, выявление повторяющихся ключей и формирование итоговой модели `CommandLineModel`. Исходный код функции представлен в листинге 1.

Листинг 1 – Код функции

```
public static ParseResult ParseArguments(string[] tokens)
{
    if (tokens == null)
        return ParseResult.Fail("Аргументы командной строки не заданы.");

    string command = tokens[0];

    if (string.IsNullOrEmpty(command))
        return ParseResult.Fail("Имя команды не задано.");

    var options = new Dictionary<string, string>();
    int i = 1;

    while (i < tokens.Length)
    {
        string current = tokens[i];

        if (!current.StartsWith("--"))
            return ParseResult.Fail($"Некорректный аргумент '{current}'. Ожидалась опция.");

        if (options.ContainsKey(current))
            return ParseResult.Fail($"Опция '{current}' указана повторно.");
    }
}
```

```

bool hasValue =
    i + 1 < tokens.Length &&
    !string.IsNullOrEmpty(tokens[i + 1]) &&
    !tokens[i + 1].StartsWith("--");

if (hasValue)
{
    options[current] = tokens[i + 1];

    i += 2;
}
else
{
    options[current] = "true";

    i++;
}
}

var model = new CommandLineModel
{
    Command = command,
    Options = options
};

return ParseResult.Success(model);
}

```

Для вычисления данной метрики требуется посчитать число уникальных операторов ($n1$) и операндов ($n2$), а также общее количество операторов ($N1$) и операндов ($N2$).

Операторы функции ParseArguments и число их вхождений приведены в таблице 6.

Таблица 6 – Операторы функции ParseArguments

№	Оператор	Число вхождений
1	public	1
2	static	1
3	if	5
4	while	1
5	return	5
6	var	2
7	new	2

Продолжение таблицы 6

№	Оператор	Число вхождений
8	=	10
9	==	1
10	!	3
11	<	2
12	&&	2
13	+=	1
14	++	1
15	.	12
16	(	18
17	)	18
18	{	7
19	}	7
20	;	15
21	,	2
22	ParseArguments	1
23	Fail	4
24	IsNullOrWhiteSpace	2
25	StartsWith	2
26	ContainsKey	1
27	Success	1

Общее число операторов составляет $N_1 = 127$ при $n_1 = 27$ уникальных операторах. Операнды функции ParseArguments и число их вхождений приведены в таблице 7.

Таблица 7 – Операнды функции ParseArguments

№	Операнд	Число вхождений
1	tokens	9
2	command	3
3	options	5
4	i	9
5	current	5

Продолжение таблицы 7

№	Операнд	Число вхождений
6	hasValue	2
7	model	2
8	Command	1
9	Options	1
10	string	3
11	Dictionary<string, string>	1
12	CommandLineModel	1
13	0	1
14	1	5
15	2	1
16	«--»	2
17	«true»	1

Общее число операндов составляет $N_2 = 52$ при $n_2 = 17$ уникальных операндах. По полученным значениям рассчитаны производные характеристики программы по метрике Холстеда:

Словарь программы вычисляется по формуле:

$$n = n_1 + n_2 = 27 + 17 = 44.$$

Длина программы вычисляется по формуле:

$$N = N_1 + N_2 = 127 + 52 = 179.$$

Объём программы вычисляется по формуле:

$$V = N \cdot \log_2 n \approx 977.$$

Сложность программы вычисляется по формуле:

$$D = \left(\frac{n_1}{2}\right) \cdot \left(\frac{N_2}{n_2}\right) \approx 41.$$

Оценка интеллектуальных усилий вычисляется по формуле:

$$E = D \cdot V \approx 40354.$$

Рассчитанные показатели позволяют охарактеризовать функцию ParseArguments как обладающую умеренным объёмом и достаточно плотной логической структурой. Это объясняется наличием нескольких проверок входных данных, обработкой опций командной строки и контролем корректности

их задания в пределах одной функции. Значения сложности и оценки интеллектуальных усилий указывают на то, что для понимания и изменения функции требуются определённые затраты, однако они остаются в пределах, допускающих дальнейшее сопровождение кода без обязательной декомпозиции.

Таким образом, расчёт метрик Холстеда показал, что функция ParseArguments имеет умеренный объём и приемлемый уровень логической сложности. Полученные результаты позволяют сделать вывод о достаточной читаемости и сопровождаемости реализации, при том что основная сложность кода обусловлена необходимостью проверки входных параметров и обработки различных вариантов задания аргументов командной строки.

5.4.2 Метрики Чепина

Метрика Чепина используется для оценки сложности программного модуля с точки зрения характера использования переменных [22]. В отличие от метрик, основанных на объёме кода или составе операторов, данный подход позволяет оценить структуру реализации через роли переменных в вычислениях и управлении потоком выполнения. Это даёт возможность судить о сложности функции с позиции организации данных и логики их обработки. Расчёт выполняется для функции ParseArguments, использованной ранее при анализе по метрике Холстеда.

В рамках метрики Чепина все переменные, присутствующие в функции, распределяются по четырём функциональным группам в зависимости от их роли в работе модуля. К группе Р относятся вводимые переменные, используемые при вычислениях и формировании результата. Группу М составляют переменные, значения которых изменяются в ходе выполнения функции. В группу С включаются управляющие переменные, определяющие ветвление и порядок выполнения операций. Группа Т объединяет паразитные переменные, не участвующие в вычислениях и не влияющие на логику работы. При этом одна и та же переменная может быть отнесена сразу к нескольким группам, если в пределах функции она выполняет несколько ролей.

Метрика Чепина рассчитывается по формуле:

$$Q = a1 \cdot P + a2 \cdot M + a3 \cdot C + a4 \cdot T,$$

где $a1$, $a2$, $a3$, $a4$ – весовые коэффициенты, отражающие степень влияния каждой группы переменных на сложность модуля. Наибольший вес присваивается управляющим переменным, так как именно они в наибольшей степени влияют на ход выполнения программы. В расчёте используются коэффициенты $a1 = 1$, $a2 = 2$, $a3 = 3$, $a4 = 0,5$.

Для функции `ParseArguments` были выделены следующие группы переменных:

- вводимые переменные: `tokens`, `command`, `current`;
- модифицируемые переменные: `command`, `options`, `i`, `current`, `hasValue`, `model`;
- управляющие переменные: `tokens`, `i`, `current`, `hasValue`, `command`;
- паразитные переменные: отсутствуют.

Подстановка полученных значений в формулу даёт:

$$Q = 1 \cdot 3 + 2 \cdot 6 + 3 \cdot 5 + 0,5 \cdot 0 = 30.$$

Полученное значение метрики Чепина показывает, что функция `ParseArguments` обладает умеренной сложностью с точки зрения работы с переменными. Основной вклад в итоговую оценку вносят управляющие и модифицируемые переменные, что связано с последовательной проверкой входных данных, обработкой опций командной строки и наличием условных переходов, определяющих порядок разбора аргументов. Это соответствует назначению функции, реализующей не отдельное простое действие, а последовательную процедуру анализа, проверки и преобразования входных параметров в итоговую модель данных.

Заключение

В рамках выпускной квалификационной работы было разработано программное средство для работы с электронными сообщениями защищённого электронного документооборота.

На первом этапе был проведён анализ предметной области: рассмотрено, как электронные сообщения служат основным средством обмена документами, служебной информацией и сопутствующими данными, и какие требования предъявляются к их представлению, обработке и защите. Это дало возможность сформулировать требования к программному средству и наметить основные направления его проектирования.

Был выполнен и функционально-стоимостной анализ, по итогам которого удалось оценить трудоёмкость разработки и определить ориентировочную стоимость реализации проекта.

На этапе проектирования была выстроена архитектура программного средства, выделены его основные модули и определено, как они взаимодействуют между собой. Чтобы наглядно описать систему, были построены диаграммы, отражающие структуру приложения, его основные сценарии использования и внутреннее устройство ключевых компонентов.

В практической части были реализованы средства формирования, передачи, получения и проверки электронных сообщений, а также механизмы интеграции с внешними средствами обработки сообщений и формирования электронной подписи. Готовое приложение работает в терминальной среде, принимает параметры из разных источников, позволяет настраивать своё поведение через конфигурацию и выполняет основные сценарии работы с электронными сообщениями.

Работоспособность программного средства подтвердили функциональное, модульное и интеграционное тестирование. Проверки показали, что основные функции приложения реализованы корректно, а его компоненты согласованно работают как при выполнении отдельных команд, так и в рамках целостных сценариев обработки сообщений. Достоверность разработки подтверждается

актом о внедрении, выданным компанией «ИнфоТеКС», что зафиксировано актом о внедрении в приложении Е.

В итоге цель выпускной квалификационной работы достигнута, а все основные задачи выполнены: анализ предметной области, проектирование, реализация и тестирование программного средства.

Список использованных источников

1. ГОСТ Р ИСО 15489-1-2019. Система стандартов по информации, библиотечному и издательскому делу. Информация и документация. Управление документами. Ч. 1. Понятия и принципы [Электронный ресурс]. URL: <https://docs.cntd.ru/document/1200163564> (дата обращения: 10.05.2026).
2. Федеральный закон от 27.07.2006 № 149-ФЗ «Об информации, информационных технологиях и о защите информации» [Электронный ресурс]. URL: <https://docs.cntd.ru/document/901990051> (дата обращения: 11.05.2026).
3. ГОСТ Р 54471-2011/ISO/TR 15801:2009. Системы электронного документооборота. Управление документацией. Информация, сохраняемая в электронном виде. Рекомендации по обеспечению достоверности и надежности [Электронный ресурс]. URL: <https://docs.cntd.ru/document/1200093863> (дата обращения: 11.05.2026).
4. Федеральный закон от 06.04.2011 № 63-ФЗ «Об электронной подписи» [Электронный ресурс]. URL: <https://docs.cntd.ru/document/902271495> (дата обращения: 12.05.2026).
5. RFC 3029. Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocol [Электронный ресурс]. URL: <https://www.rfc-editor.org/rfc/rfc3029.html> (дата обращения: 13.05.2026).
6. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) [Электронный ресурс]. URL: <https://www.w3.org/TR/soap12-part1> (дата обращения: 15.05.2026).
7. RFC 3902. The «application/soap+xml» media type [Электронный ресурс]. URL: <https://www.rfc-editor.org/rfc/rfc3902> (дата обращения: 16.05.2026).
8. RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies [Электронный ресурс]. URL: <https://www.rfc-editor.org/info/rfc2045/> (дата обращения: 17.05.2026).
9. RFC 2046. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types [Электронный ресурс]. URL: <https://www.rfc-editor.org/info/rfc2046/> (дата обращения: 17.05.2026).

10. ETSI EN 319 132-1. Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures [Электронный ресурс]. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31913201/01.03.01_60/en_31913201v010301p.pdf (дата обращения: 18.05.2026).

11. ETSI EN 319 122-1. Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures [Электронный ресурс]. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31912201/01.03.01_60/en_31912201v010301p.pdf (дата обращения: 17.05.2026).

12. ETSI EN 319 142-1. Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures [Электронный ресурс]. URL: https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.02.01_60/en_31914201v010201p.pdf (дата обращения: 18.05.2026).

13. Шеремет, А. Д. Функционально-стоимостной анализ: учебное пособие / А. Д. Шеремет, А. П. Ковалёв ; под ред. А. Д. Шеремета. – М.: Экономический факультет МГУ имени М. В. Ломоносова, 2017. – 204 с.

14. HeadHunter. Динамика ожидаемых и предлагаемых зарплат [Электронный ресурс]. URL: https://stats.hh.ru/penza_oblast (дата обращения: 18.05.2026).

15. Нотация IDEF0 [Электронный ресурс]. URL: https://www.businessstudio.ru/upload/files/notation_idef0.pdf (дата обращения: 19.05.2026).

16. Диаграмма вариантов использования // Национальный открытый университет ИНТУИТ [Электронный ресурс]. URL: <http://www.intuit.ru/studies/courses/32/32/lecture/1004> (дата обращения: 20.05.2026).

17. Буч, Г. Язык UML. Руководство пользователя: пер. с англ. / Н. Мухин. – 2-е изд. – Москва: ДМК Пресс, 2006. – 496 с.

18. Архитектурные принципы // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/architectural-principles> (дата обращения: 22.05.2026).

19. Настройка в .NET // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/core/extensions/configuration> (дата обращения: 24.05.2026).
20. Крауч, Р. Тестирование ПО. Основы, методы и термины / Р. Крауч, В. Д. Влиет. – СПб. : Питер, 2003. – 256 с.
21. Методы черного, белого и серого ящиков: в чем разница? // SkillFactory Media [Электронный ресурс]. URL: <https://blog.skillfactory.ru/metodi-chnernogo-belogo-serogo-yashikov/> (дата обращения: 25.05.2026).
22. Метрики как средства управления качеством // Открытые системы [Электронный ресурс]. URL: <https://www.osp.ru/os/2009/08/10748698> (дата обращения: 26.05.2026).

Приложение А
(справочное)
Глоссарий

B2G (Business-to-Government) – формат или тип взаимодействия, при котором обмен сообщениями осуществляется между бизнес-структурой и государственным органом.

G2G (Government-to-Government) – формат или тип взаимодействия, при котором обмен сообщениями осуществляется между государственными органами или государственными информационными системами.

ЭДО (электронный документооборот) – обмен документами и связанными данными в электронной форме.

Электронное сообщение – структурированный набор данных для передачи документов и служебной информации.

SOAP (Simple Object Access Protocol) – формат XML-сообщений для межсистемного обмена данными.

MIME (Multipurpose Internet Mail Extensions) – формат составного сообщения, позволяющий передавать данные и вложения в одном контейнере.

XML (eXtensible Markup Language) – язык разметки для структурированного представления данных.

ЭП (электронная подпись) – данные, присоединённые к электронному документу и используемые для подтверждения его происхождения.

XAdES (XML Advanced Electronic Signatures) – формат электронной подписи для XML-документов и XML-сообщений.

CAdES (CMS Advanced Electronic Signatures) – формат электронной подписи для криптографического контейнера подписанных данных.

PAdES (PDF Advanced Electronic Signatures) – формат электронной подписи для PDF-документов.

Хеширование – вычисление хеш-значения данных для контроля их целостности.

CLI (Command Line Interface) – интерфейс взаимодействия с программой через командную строку.

JSON (JavaScript Object Notation) – текстовый формат представления структурированных данных.

DTO (Data Transfer Object) – объект передачи данных между частями системы.

Конфигурация приложения – набор параметров, определяющих режимы работы программы.

Конвейер обработки – последовательность этапов подготовки и преобразования данных.

Приложение Б
(обязательное)
Исходный код приложения

Листинг Б.1 – AttachmentManagerTests.cs

```

/// <summary>
/// Тесты для <see cref="AttachmentManager"/>.
/// </summary>
public class AttachmentManagerTests
{
    /// <summary>
    /// Проверяет, что метод <see
    cref="AttachmentManager.ProcessSize(string)"/>
    /// корректно обрабатывает различные варианты размера.
    /// </summary>
    [Theory]
    [InlineData("150", 150)]
    [InlineData("150B", 150)]
    [InlineData("100KB", 102400)]
    [InlineData("150kb", 153600)]
    [InlineData("1MB", 1048576)]
    [InlineData("15mb", 15728640)]
    public void ProcessSize_ListData_Success(string inputSize,
    long resultSize)
    {
        // Действие
        var result = _manager.ProcessSize(inputSize);

        // Проверка
        Assert.Equal(result, resultSize);
    }

    /// <summary>
    /// Проверяет, что метод <see
    cref="AttachmentManager.GetAttachment(MessageAttachment, bool)"/>
    /// корректно формирует вложение из случайных байтовых данных.
    /// </summary>
    [Fact]
    public void GetAttachment_RandomBytes_Success()
    {
        // Подготовка
        const string SizeAttachment = "10MB";
        const int CountAttachment = 4;
        var data = new MessageAttachment.RandomBytes(
            Size: SizeAttachment,
            Count: CountAttachment
        );

        // Действие
        var result = _manager.GetAttachment(data);

        // Проверка
        ValidateAttachment(result, SizeAttachment);
    }
}

```

```

    /// <summary>
    /// Проверяет, что метод <see
    cref="AttachmentManager.GetAttachment(MessageAttachment, bool)"/>
    /// корректно формирует PDF вложение из случайных байтовых
    данных.
    /// </summary>
    [Fact]
    public void GetAttachment_RandomPdfBytes_Success()
    {
        // Подготовка
        const string SizeAttachment = "20MB";
        var data = new MessageAttachment.RandomPdfBytes(
            Size: SizeAttachment
        );

        // Действие
        var result = _manager.GetB2gAttachment(data);

        // Проверка
        ValidateAttachment(result, SizeAttachment);
    }

    /// <summary>
    /// Проверяет, что метод <see
    cref="AttachmentManager.GetAttachment(MessageAttachment, bool)"/>
    /// корректно формирует вложение из случайных байтовых данных.
    /// </summary>
    [Fact]
    public void GetG2gAttachment_RandomBytes_Success()
    {
        // Подготовка
        const string SizeAttachment = "10MB";
        const int CountAttachment = 4;
        const bool IsMime = false;
        var data = new MessageAttachment.RandomBytes(
            Size: SizeAttachment,
            Count: CountAttachment
        );

        // Действие
        var result = _manager.GetAttachment(data, IsMime);

        // Проверка
        ValidateAttachment(result, SizeAttachment);
    }

    private void ValidateAttachment(G2gAttachmentData
    attachmentData, string size)
    {
        foreach (var attachment in attachmentData.Attachments)
        {
            ValidateAttachment(attachment, size,
            attachmentData.Attachments.Count());
        }
    }

```

```

    }
    foreach (var fileName in attachmentData.FileNames)
    {
        ValidateValue(fileName);
    }
    foreach (var mediaType in attachmentData.MediaTypees)
    {
        ValidateValue(mediaType);
    }
    foreach (var id in attachmentData.AttachmentIds)
    {
        ValidateValue(id);
    }
}

private void ValidateAttachment(byte[] attachment, string
stringSize, int countAttachment = 1)
{
    var size = _manager.ProcessSize(stringSize) /
countAttachment;

    Assert.NotNull(attachment);
    Assert.NotEmpty(attachment);
    Assert.Contains(attachment, b => b != 0);
    Assert.True(attachment.Length >= size);
}
private void ValidateValue(string? value)
{
    Assert.NotNull(value);
    Assert.NotEmpty(value);
}

private static Mock<ITracer> _tracerMock = new
Mock<ITracer>();
private AttachmentManager _manager = new
AttachmentManager(_tracerMock.Object);
}

```

Листинг Б.2 – ParserTests.cs

```

/// <summary>
/// Тесты для <see cref="Parser"/>.
/// </summary>
public class ParserTests
{
    /// <summary>
    /// Проверяет, что метод <see cref="Parser.ParseArgs(string[],
Dictionary{string, Command})"/>
    /// возвращает исключение при отсуствие аргументов командной
строки.
    /// </summary>
    [Fact]

```

```

public void ParseArgs_EmptyArgs_ParseArgsException()
{
    // Подготовка
    var app = GetCliApp();
    var args = new string[] { };

    // Действие
    var ex = Assert.Throws<ParserArgsException>(() =>
Parser.ParseArgs(args, app.Commands));

    // Проверка
    Assert.Equal("Аргументы командной строки отсутствуют.",
ex.Message);
}

/// <summary>
/// Проверяет, что метод <see cref="Parser.ParseArgs(string[],
Dictionary{string, Command})"/>
/// возвращает исключение при вызове несуществующей команды.
/// </summary>
[Fact]
public void ParseArgs_FakeCommand_ParseArgsException()
{
    // Подготовка
    var fakeToken = "fake-command";
    var app = GetCliApp();
    var args = new string[] { fakeToken };

    // Действие
    var ex = Assert.Throws<ParserArgsException>(() =>
Parser.ParseArgs(args, app.Commands));

    // Проверка
    Assert.Equal($"Данной команды не существует: {fakeToken}",
ex.Message);
}

/// <summary>
/// Проверяет, что метод <see cref="Parser.ParseArgs(string[],
Dictionary{string, Command})"/>
/// корректно обрабатывает вызов корневой справки.
/// </summary>
/// <param name="helpToken">Варианты флага справки.</param>
[Theory]
[InlineData("-h")]
[InlineData("--help")]
public void ParseArgs_HelpFlag_Success(string helpToken)
{
    // Подготовка
    var app = GetCliApp();
    var args = new string[] { helpToken };

    // Действие

```

```

        var result = Parser.ParseArgs(args, app.Commands);

        // Проверка
        Assert.Null(result.CurrentCommand);
        Assert.False(result.IsPrintCommandHelp);
        Assert.True(result.IsPrintRootHelp);
    }

    /// <summary>
    /// Проверяет, что метод <see cref="Parser.ParseArgs(string[],
Dictionary{string, Command})"/>
    /// корректно обрабатывает вызов справки по команде.
    /// </summary>
    /// <param name="helpToken">Варианты флага справки.</param>
    [Theory]
    [InlineData("-h")]
    [InlineData("--help")]
    public void ParseArgs_HelpFlagCommand_Success(string
helpToken)
    {
        // Подготовка
        var commandToken = "send-message";
        var app = GetCliApp();
        var args = new string[] { commandToken, helpToken };

        // Действие
        var result = Parser.ParseArgs(args, app.Commands);

        // Проверка
        Assert.NotNull(result.CurrentCommand);
        Assert.True(result.IsPrintCommandHelp);
        Assert.False(result.IsPrintRootHelp);
    }

    /// <summary>
    /// Проверяет, что метод <see cref="Parser.ParseArgs(string[],
Dictionary{string, Command})"/>
    /// корректно обрабатывает вызов справки по команде.
    /// </summary>
    [Fact]
    public void ParseArgs_StatusCryptoCommand_Success()
    {
        // Подготовка
        var alias = "-ll";
        var commandToken = "status-crypto";
        var app = GetCliApp();
        var args = new string[] { commandToken, alias, "0" };

        // Действие
        var result = Parser.ParseArgs(args, app.Commands);

        // Проверка
        Assert.NotNull(result.CurrentCommand);

```

```

        Assert.False(result.IsPrintCommandHelp);
        Assert.False(result.IsPrintRootHelp);

        Assert.Equal(result.CurrentCommand.Name, commandToken);
        Assert.NotNull(result.CurrentCommand.Description);
        Assert.NotNull(result.CurrentCommand.HelpInfo);

Assert.NotNull(result.CurrentCommand.Options.First().Alias);
    }

    private CliApp GetCliApp()
    {
        var provider = ServiceManager.ConfigureServices(new
AppOptions(new TracerParameters(), false));
        CliApp app = provider.GetRequiredService<CliApp>();
        CommandsRegister register =
provider.GetRequiredService<CommandsRegister>();
        register.Register(app);

        return app;
    }
}

```

Приложение В
(рекомендуемое)
Диаграммы классов

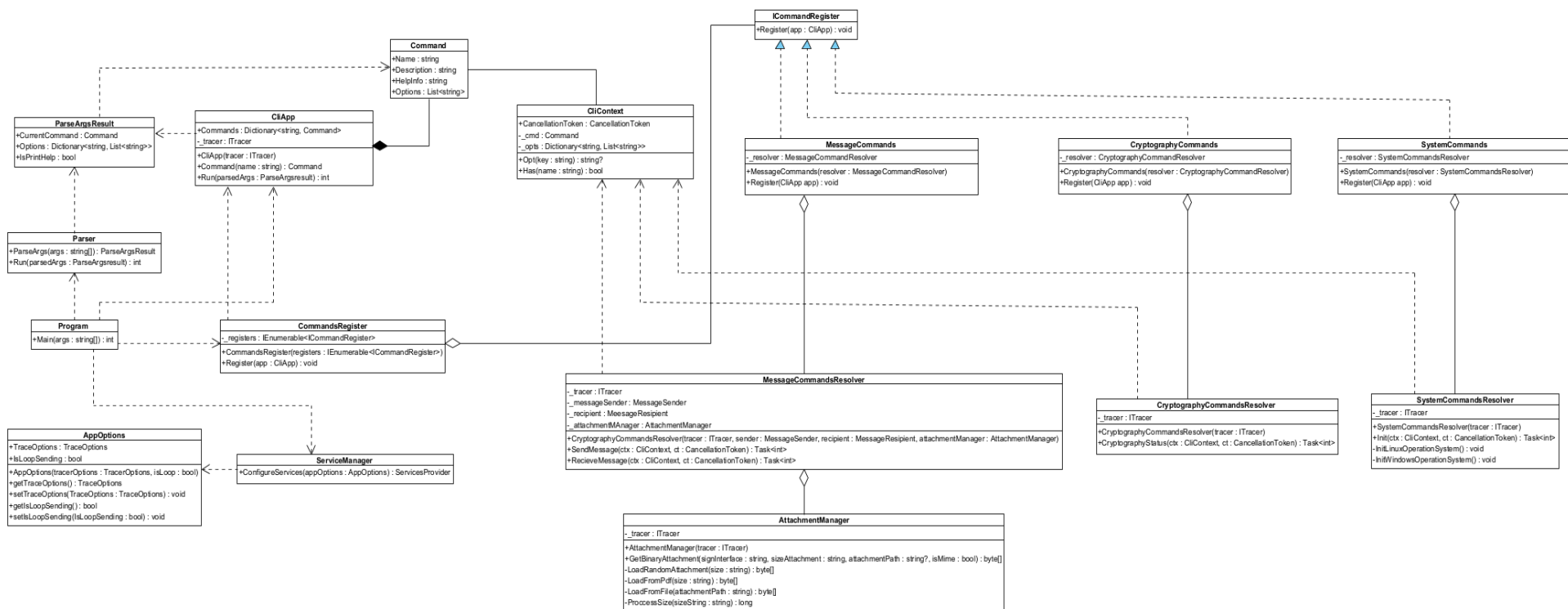


Рисунок В.1 – Диаграмма классов главного модуля

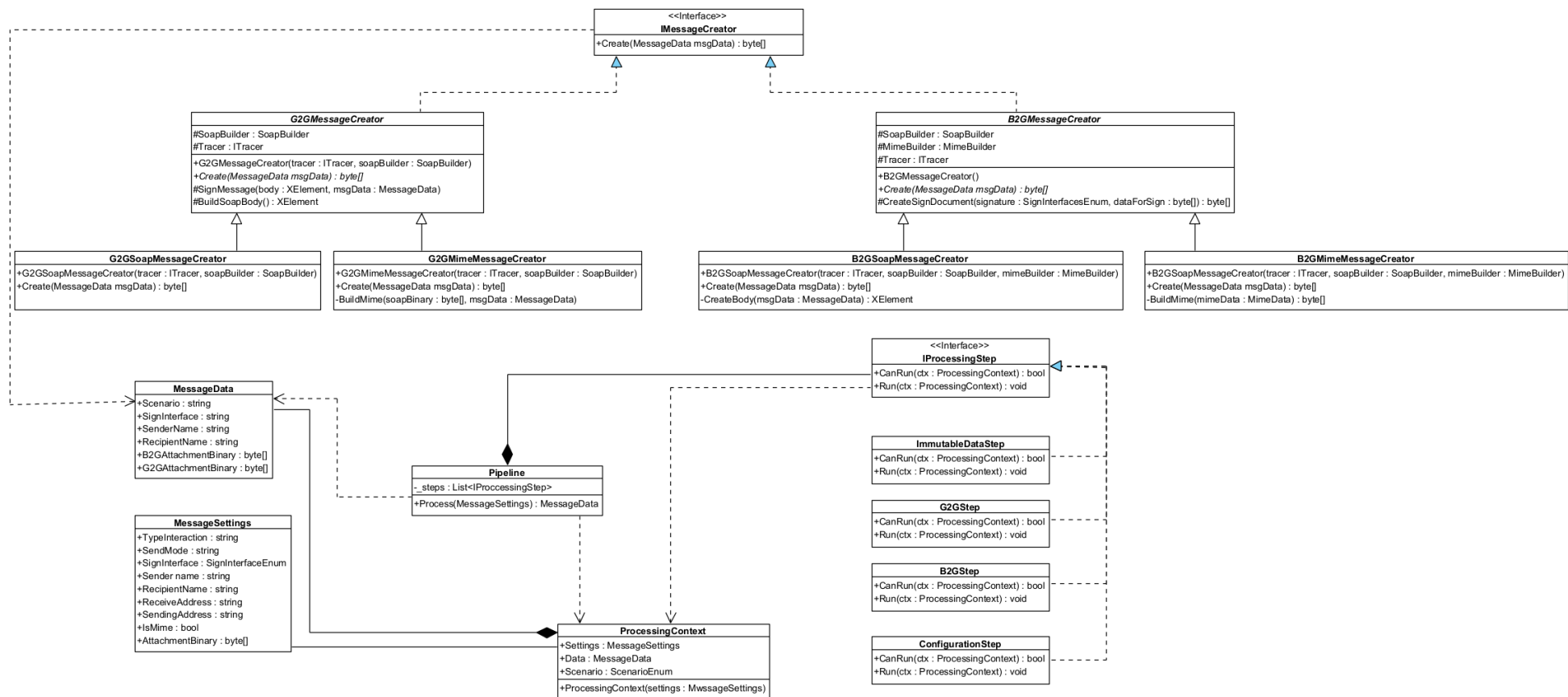


Рисунок В.2 – Диаграмма классов фабрики создания сообщений

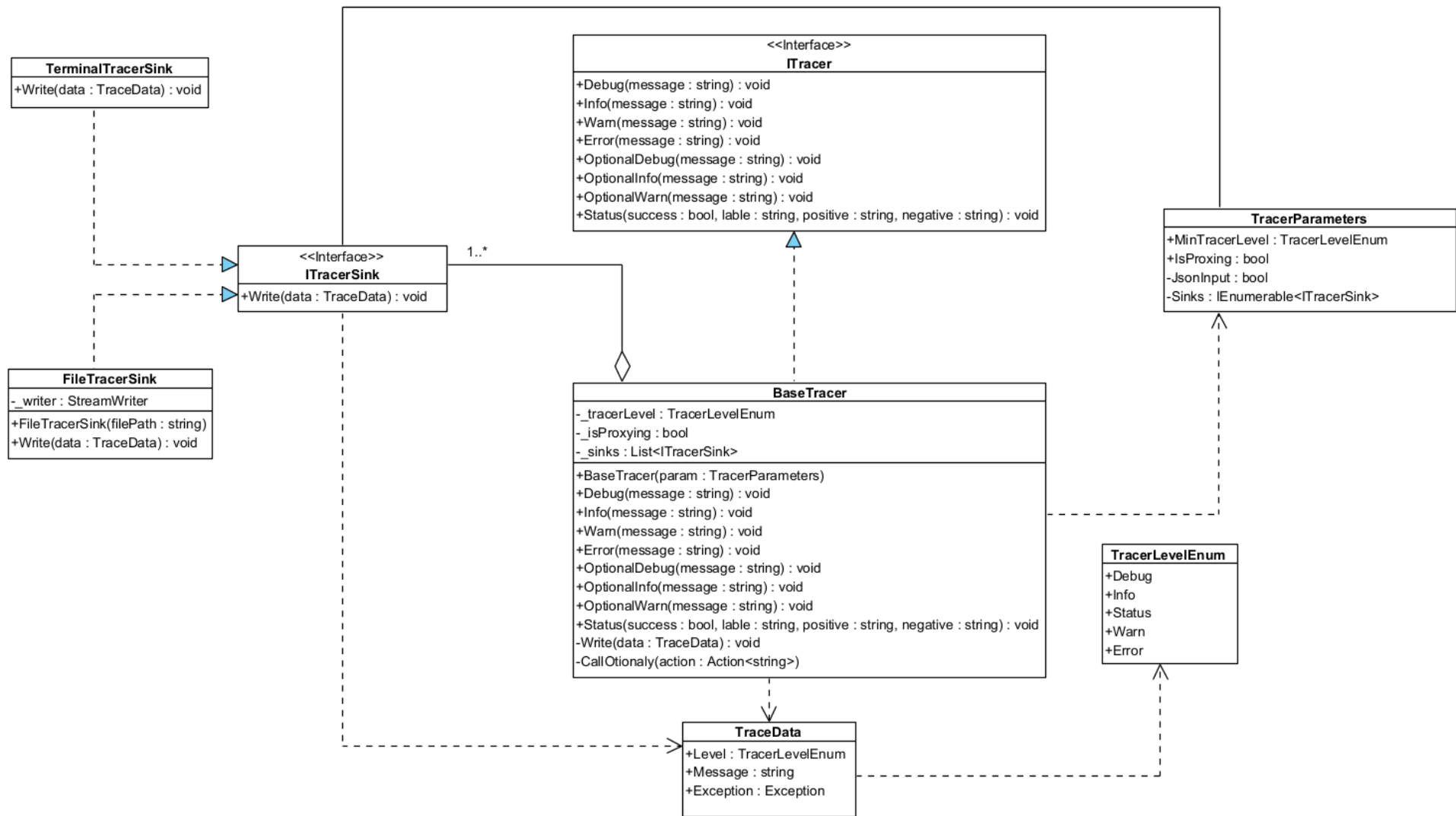


Рисунок В.3 – Диаграмма классов библиотеки ведения записей

Приложение Г
(рекомендуемое)
Результаты функционального тестирования

```

12345:~# emulator send-message --mode AutoSend --mime
[19:30:12] [ info ] Документ успешно создан службой: G2G.G2GMimeMessageCreator в автоматическом режиме.
[19:30:17] [ status ] Статус очередей сегмента Gateway-Emulator: [Подключены]
[19:30:17] [ result ] ok
[SendMessageResult]
[data] Старт операции : 19:30:12
[data] Конец операции : 19:30:17
[data] Id сообщения : urn:uuid:470233e0-8f96-40f6-919b-fbe682fa261a
12345:~#

```

Рисунок Г.1 – Результат автоматического создание и отправки сообщения в MIME-контейнере

```

12345:~# emulator send-message --mode Create --outputpath /home/test-msg --sign-interface XAdES
[19:28:08] [ info ] Документ успешно создан службой: G2G.G2GSoapMessageCreator в автоматическом режиме.
[19:28:08] [ result ] ok
[SendMessageResult]
[data] Старт операции : 19:28:08
[data] Конец операции : 19:28:08
[data] Id сообщения : urn:uuid:b74c1d59-7257-435e-b7cc-dc645f3615b2
12345:~# emulator send-message --mode Create --outputpath /home/test-msg --sign-interface CAdES --type B2G
[19:28:09] [ info ] Документ успешно создан службой: B2G.B2GSoapMessageCreator в автоматическом режиме.
[19:28:09] [ result ] ok
[SendMessageResult]
[data] Старт операции : 19:28:09
[data] Конец операции : 19:28:09
[data] Id сообщения : urn:uuid:5a612ce7-5f43-445f-b026-40ef0b1c3035
12345:~# emulator send-message --mode Create --outputpath /home/test-msg --sign-interface PAdES --type B2G
[19:28:22] [ info ] Документ успешно создан службой: B2G.B2GSoapMessageCreator в автоматическом режиме.
[19:28:22] [ result ] ok
[SendMessageResult]
[data] Старт операции : 19:28:22
[data] Конец операции : 19:28:22
[data] Id сообщения : urn:uuid:eed6a180-9c20-4d1c-b851-0e9028dca455
12345:~#

```

Рисунок Г.2 – Результат создания сообщения с разными видами формата ЭП

```

12345:~# emulator _init-emulator -ll 0
[19:14:04] [ debug ] Старт InitEmulator

[19:14:04] [ info ] Система определена как Linux ОС.
[19:14:04] [ debug ] Обработка auto-complete скрипта.
[19:14:04] [ debug ] Скрипт установлен в /etc/bash_completion.d/emulator-prompt
[19:14:04] [ info ] Auto-complete скрипт инициализирован.
[19:14:04] [ debug ] Обработка dsmqver скрипта.
[19:14:04] [ debug ] Скрипт установлен в /usr/bin/dspmqrver
[19:14:04] [ info ] Dsmqver скрипт инициализирован.
[19:14:04] [ debug ] Обработка alias скрипта.
[19:14:04] [ debug ] Скрипт установлен в /usr/bin/emulator
[19:14:04] [ info ] Alias скрипт инициализирован.
[19:14:04] [ debug ] Конец InitEmulator

[19:14:04] [ info ] Инициализация эмулятора на Linux ОС завершена.
12345:~#

```

Рисунок Г.3 – Результат инициализации программного средства

```

12345:~# emulator send-message --mode Create --outputpath /home/test-msg
[19:25:02] [ info ] Документ успешно создан службой: G2G.G2GSoapMessageCreator в автоматическом режиме.
[19:25:02] [ result ] ok
[SendMessageResult]
[data] Старт операции : 19:25:02
[data] Конец операции : 19:25:02
[data] Id сообщения : urn:uuid:5c596239-f758-4271-a32f-2deaca790a45

```

Рисунок Г.4 – Результат создания сообщения с настройками по умолчанию

```

12345:~# emulator send-message --mode Create --outputpath /home/test-msg -ll 0 --size 15MB
[19:25:59] [ debug ] Старт отправки сообщения в режиме G2G.
[19:25:59] [ debug ] Размер конвертирован : 15MB -> 15728640 байт.
[19:25:59] [ debug ] Старт получения криптографии национального стандарта.
[19:25:59] [ debug ] Серийный номер сертификата: A30B9AC8A
[19:25:59] [ debug ] Конец получения криптографии национального стандарта.
[19:25:59] [ debug ] Количество сертификатов, использованных для подписи: 1.
[19:26:00] [ info ] Документ успешно создан службой: G2G.G2GSoapMessageCreator в автоматическом режиме.
[19:26:00] [ debug ] Сформированное сообщение было сохранено в /home/test-msg.
[19:26:00] [ debug ] Сценарий генерации данных сообщения: GatewayG2GMsg
[19:26:00] [ debug ] Данные генерации сообщения:
[MessageData]
[data] SenderSegmentName : senderName
[data] RecipientSegmentName : recipientName
[SoapHeader]
[data] To : addressTo
[data] From : null
[data] RelatesTo : null
[data] RelatesActionAttr : null
[data] ReplyTo : addressReplyTo
[data] Action : int://CP/P.
[data] TrackId : urn:uuid:e83b0936-9367-48d6-92a6-b866b58c2ed6
[data] AcceptTime : 2024-12-24T06:27:51
[data] ConversationId : urn:uuid:5b21f902-7ee2-4efc-acbf-4e69d954c245
[data] ProcedureId : urn:uuid:efae83b8-78a5-49bf-bcc8-aedb4a0e5815
[data] FaultTo : null
[data] MessageId : urn:uuid:08b5ef34-2f9c-4d46-b9d1-33ae41c22e57
[MimeProperties]
[data] Boundary : null
[data] Type : null
[data] ContentType : null
[data] MimeVersion : null
[data] StartId : null
[data] StartInfo : null
[19:26:00] [ result ] ok
[SendMessageResult]
[data] Старт операции : 19:25:59
[data] Конец операции : 19:26:00
[data] Id сообщения : urn:uuid:08b5ef34-2f9c-4d46-b9d1-33ae41c22e57
[19:26:00] [ debug ] Конец отправки сообщения в режиме G2G.

```

Рисунок Г.5 – Результат создания сообщения с настройкой вложения

```

12345:~# emulator send-message --mode CompleteMessageSend --inputpath /home/test-msg -ll 0
[19:24:02] [ debug ] Старт отправки сообщения в режиме G2G.
[19:24:02] [ info ] Сообщение успешно прочитано из файла: /home/test-msg
[19:24:02] [ debug ] Старт проверки подключения к очереди сегмента: Gateway-Emulator.
[19:24:02] [ status ] Статус очередей сегмента Gateway-Emulator: [Подключены]
[19:24:02] [ debug ] Конец проверки подключения к очереди: Gateway-Emulator.
[19:24:02] [ result ] ok
[SendMessageResult]
[data] Старт операции : 19:24:02
[data] Конец операции : 19:24:02
[data] Id сообщения : d80adae5-5d11-40fa-b008-396c70766887
[19:24:02] [ debug ] Конец отправки сообщения в режиме G2G.
12345:~#

```

Рисунок Г.6 – Результат отправки готового сообщения

```

12345:~# emulator status-crypto
[19:12:29] [ result ] ok
[ClearQueueResult]
[data] Национальный стандарт криптографии : Доступен
[data] Трансграничный стандарт криптографии : Доступен
12345:~# emulator status-crypto -ll 0
[19:12:35] [ debug ] Старт получения криптографии национального стандарта.
[19:12:35] [ debug ] Серийный номер сертификата: A30B9AC8A6
[19:12:35] [ debug ] Конец получения криптографии национального стандарта.
[19:12:35] [ debug ] Старт получения криптографии трансграничного стандарта.
[19:12:35] [ debug ] Конец получения криптографии трансграничного стандарта.
[19:12:35] [ debug ] Национальный стандарт криптографии: [Доступен]
[19:12:35] [ debug ] Трансграничный стандарт криптографии: [Доступен]
[19:12:35] [ result ] ok
[ClearQueueResult]
[data] Национальный стандарт криптографии : Доступен
[data] Трансграничный стандарт криптографии : Доступен
12345:~#

```

Рисунок Г.7 – Результат получения статуса внешней системы формирования ЭП

```

12345:~# emulator status-queue -ll 0
[11:39:13] [ debug ] Старт операции: QueueStatus
[11:39:13] [ debug ] Старт проверки подключения к очереди сегмента: Gateway-Emulator.
[11:39:13] [ status ] Статус очередей сегмента Gateway-Emulator: [Подключены]
[11:39:13] [ debug ] Конец проверки подключения к очереди: Gateway-Emulator.
[11:39:13] [ result ] ok
[StatusQueueResult]
[data] Очереди сегмента -Gateway-Emulator : Доступны
[11:39:13] [ debug ] Завершение операции: QueueStatus
12345:~#

```

Рисунок Г.8 – Результат получения статуса внешней системы обработки сообщений

```

12345:~# emulator send-message --mode CompleteMessageSend
[11:41:10] [ error ] Некорректные параметры пути к сообщению для режима CompleteMessageSend

```

Рисунок Г.9 – Результат корректной обработки отсутствия обязательного аргумента команды

```

12345:~# emulator send-message --mode AutoSend --size 150MB
[11:42:27] [ error ] Размер вложения 157286400 байт не соответствует доступному размеру (1 < размер вложения < 78255226 байт)

```

Рисунок Г.10 – Результат корректной обработки превышения допустимого значения аргумента команды

```

12345:~# emulator receive-message --segmentName Gateway-Emulator --queue-name OUT.QUEUE --outputpath /home/msg -ll 0
[18:59:08] [ debug ] Для команды задан таймаут: 10000 мс.
[18:59:08] [ debug ] Старт получения сообщения.
[18:59:08] [ debug ] Старт проверки подключения к очереди сегмента: Gateway-Emulator.
[18:59:08] [ status ] Статус очередей сегмента Gateway-Emulator: [Подключены]
[18:59:08] [ debug ] Конец проверки подключения к очереди: Gateway-Emulator.
[18:59:08] [ info ] Количество сообщений в очереди OUT.QUEUE: 2.
[18:59:08] [ debug ] Тип сообщения определен как SoapXml.
[18:59:08] [ debug ] Роль сообщения: SoapFault
[18:59:08] [ info ] Сообщение успешно получено и сохранено: /home/msg.
[18:59:08] [ debug ] Конец получения сообщения.

```

Рисунок Г.11 – Результат выполнения команды получения сообщения

```

12345:~# emulator send-message --loop
[18:59:46] [ info ] Старт периодической отправки.
[18:59:46] [ status ] Статус очередей сегмента | -Gateway-Emulator: [Подключены]
[18:59:46] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:46
[data] Конец операции : 18:59:46
[data] Id сообщения : urn:uuid:c8e769a3-b9a3-48a7-8cda-266f67532b12
[18:59:46] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:46
[data] Конец операции : 18:59:46
[data] Id сообщения : urn:uuid:8185dfe9-9537-4fa8-b8d2-aa6120e64e7c
[18:59:47] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:46
[data] Конец операции : 18:59:47
[data] Id сообщения : urn:uuid:f1c73b2a-1e1d-4d99-9340-3ce240dc7d93
[18:59:47] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:47
[data] Конец операции : 18:59:47
[data] Id сообщения : urn:uuid:12253b54-e8ea-4431-ae2f-b94b9995369f
[18:59:47] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:47
[data] Конец операции : 18:59:47
[data] Id сообщения : urn:uuid:1920bd3b-d0d2-4ee2-a8f3-70d0951f0ac6
[18:59:57] [ status ] Статус периодической отправки: [1/2 пакет, 5/5 сообщений.]
[18:59:57] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:57
[data] Конец операции : 18:59:57
[data] Id сообщения : urn:uuid:e07444f7-870f-4a98-97df-3f928c1ef41c
[18:59:57] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:57
[data] Конец операции : 18:59:57
[data] Id сообщения : urn:uuid:3b14191c-6d3f-4e38-9f27-241e518369bc
[18:59:57] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:57
[data] Конец операции : 18:59:57
[data] Id сообщения : urn:uuid:4262849a-c348-4100-911f-cfd171ad4e27
[18:59:57] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:57
[data] Конец операции : 18:59:57
[data] Id сообщения : urn:uuid:f248db46-b8e5-4cef-8bd1-f786ae9d67c5
[18:59:57] [ result ] ok
[SendMessageResult]
[data] Старт операции : 18:59:57
[data] Конец операции : 18:59:57
[data] Id сообщения : urn:uuid:c21974a0-cd11-4000-8b91-0815ab0be54d
[18:59:57] [ status ] Статус периодической отправки: [2/2 пакет, 5/5 сообщений.]
[18:59:57] [ info ] Отправка сообщений завершена с результатом 10/10 за 00:00:10.
12345:~#

```

Рисунок Г.12 – Результат выполнения команды циклической отправки сообщений

```

PS D:\Active\Projects\ linux-x64> dotnet .\Emulator.dll _init-emulator --overwrite
[20:56:50] [ info ] система определена как windows os.
[20:56:50] [ info ] Инициализация эмулятора на Windows OS завершена.
PS D:\Active\Projects\ linux-x64> emulator status-queue --segmentName Gateway-Emulator
[20:56:56] [ status ] Статус очереди сегмента Gateway-Emulator: [Подключены]
[20:56:56] [ result ] ok
[StatusQueueResult]
[data] Очереди сегмента Gateway-Emulator : Доступны
PS D:\Active\Projects\ linux-x64>

```

Рисунок Г.13 – Результат выполнения команды на операционной системе Windows

Приложение Д
(рекомендуемое)

Листинг конфигурации системы интеграционного тестирования

```

{
  // Общие значения.
  "variables": {
    "dllPath": "Emulator.dll",
    "htmlReportPath": "/home/data/integration-report.html",
    "shortTimeout": "10",
    "defaultScenarioDelay": "300",
    "noDelay": "0",
    "fastLimit": "250",
    "mediumLimit": "2500",
    "slowLimit": "4500",
    "longLimit": "6000",
    "status_queue": "status-queue -ll 0",
    "status_crypto": "status-crypto -ll 0",
    "send-message-default": "send-message -ll 0",
    "send-message-mime": "send-message -ll 0 --direct --mime",
    "send-message-direct": "send-message -ll 0 --direct",
    "clear-queue": "clear-queue -ll 0 --segment-name Gateway-Emulator",
    "receive-message": "receive-message -ll 0 --segmentName Gateway-Emulator --queue-name OUT.QUEUE --outputpath /home/test-msg",
    "create-signdoc": "create-signdoc -ll 0",
  },
  // Управление запуском. Если поставить false, система остановится на первой ошибке.
  "run": {
    "continueOnFail": true
  },
  // Целевая терминальная программа, которую запускает система.
  "target": {
    "dllPath": "${dllPath}",
    "runnerPath": "dotnet"
  },
  "report": {
    "htmlPath": "${htmlReportPath}",
    "jsonPath": "${jsonReportPath}"
  },
  // Набор команд для тестирования.
  "commands": [
    {
      "id": "status-queue",
      "name": "Статус очередей.",
      "description": "Статус очередей сегмента.",
      "scenarioOnly": false,
      "arguments": "${status_queue}",
      "expectedExitCode": 0,
      "timeoutSeconds": "${shortTimeout}",
      "maxDurationMilliseconds": "${longLimit}"
    },
    {
      "id": "send-message",
      "name": "Отправить сообщение.",

```

```

    "description": "(по умолчанию)",
    "scenarioOnly": false,
    "arguments": "${send-message-default}",
    "expectedExitCode": 0,
    "timeoutSeconds": "${shortTimeout}",
    "maxDurationMilliseconds": "${longLimit}"
  },
  {
    "id": "send-message-direct",
    "name": "Отправить сообщение.",
    "description": "Отправка напрямую.",
    "scenarioOnly": false,
    "arguments": "${send-message-direct}",
    "expectedExitCode": 0,
    "timeoutSeconds": "${shortTimeout}",
    "maxDurationMilliseconds": "${longLimit}"
  },
  {
    "id": "send-message-mime",
    "name": "Отправить mime сообщение.",
    "description": "Отправка mime контейнера.",
    "scenarioOnly": false,
    "arguments": "${send-message-mime}",
    "expectedExitCode": 0,
    "timeoutSeconds": "${shortTimeout}",
    "maxDurationMilliseconds": "${longLimit}"
  },
  {
    "id": "clear-queue",
    "name": "Очистить очередь.",
    "description": "",
    "scenarioOnly": false,
    "arguments": "${clear-queue}",
    "expectedExitCode": 0,
    "timeoutSeconds": "${shortTimeout}",
    "maxDurationMilliseconds": "${longLimit}"
  },
  {
    "id": "receive-message",
    "name": "Получить сообщение.",
    "description": "",
    "scenarioOnly": true,
    "arguments": "${receive-message}",
    "expectedExitCode": 0,
    "timeoutSeconds": "${shortTimeout}",
    "maxDurationMilliseconds": "${longLimit}"
  }
],
// Базовый сценарий тракта.
"scenarios": [
  {
    "id": "base-flow",
    "name": "Базовый сценарий тракта сообщения.",

```

```

    "description": "",
    "commandIds": [
        "clear-queue",
        "send-message-direct",
        "receive-message"
    ],
    "delayMilliseconds": "${defaultScenarioDelay}"
},
{
    "id": "mime-flow",
    "name": "Базовый сценарий тракта mime сообщения.",
    "description": "",
    "commandIds": [
        "clear-queue",
        "send-message-mime",
        "receive-message"
    ],
    "delayMilliseconds": "${defaultScenarioDelay}"
}
]
}

```

Приложение Е
(рекомендуемое)
Акт о внедрении



Акционерное общество
«Информационные технологии
и коммуникационные системы»
(АО «ИнфоТеКС»)

Юридический адрес: ул. Мишина,
дом 56, стр. 2, этаж 2, пом. IX, комната 29
Москва, 127083

Почтовый адрес: а/я № 80,
улица Отрадная, дом 2Б, стр. 1
Москва, 127273

Тел. (495)737-61-92 Факс (495)737-72-78

ОГРН 1027739185066
ИНН/КПП 7710013769/771401001
<http://www.infotecs.ru>

Акционерное общество «Информационные
технологии и коммуникационные системы»
(АО «ИнфоТеКС»)

127083, Москва, улица Мишина, д. 56, стр. 2,
этаж 2, помещение IX, комната 29

.....08.06.2026..... №.....

Акт о внедрении

Настоящим подтверждаю, что Зиновьев Леонид Дмитриевич является сотрудником компании АО «ИнфоТеКС». Решения, описанные в выпускной квалификационной работе на тему «Базовые средства формирования электронных сообщений защищенного электронного документооборота», вошли в состав разрабатываемых технологических средств.

Выдано Зиновьеву Леониду Дмитриевичу для предъявления по требованию.

Руководитель обособленного
подразделения в г. Пенза



М.М. Грунтович