Московский государственный технический университет

им. «Н.Э.Баумана»

**Курсовая работа**

по дисциплине:

**«Технологии разработки программного обеспечения»**

на тему:

**«Разработка журнала сообщений»**

исполнители:

студенты группы ИУ5-27 (м)

Тихонов И.В. Лахвич Д. С.

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

преподаватель:

Виноградова М.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_

**Москва 2014**

Оглавление

[Варианты и распределение студентов. 3](#_Toc388731977)

[1. Этап анализа и планирования требований 4](#_Toc388731978)

[1.1 Постановка задачи. 4](#_Toc388731979)

[1.2 Спецификация основных проектных требований 4](#_Toc388731980)

[1.3 Описание бизнес процессов. 5](#_Toc388731981)

[1.4 модель предметной области 5](#_Toc388731982)

[1.5 Выявленные актеры. 7](#_Toc388731983)

[1.6 Выявленные прецеденты 7](#_Toc388731984)

[1.7 диаграмма прецедентов 7](#_Toc388731985)

[1.8 описание прецедентов 8](#_Toc388731986)

[1.9 Перечень критических рисков 9](#_Toc388731987)

[1.10 Список приоритетов прецедентов. 9](#_Toc388731988)

[1.11 Описание возможной архитектуры 9](#_Toc388731989)

[1.12 Начальная оценка стоимости, затрат и длительности проекта 10](#_Toc388731990)

[1.12 Начальный план выпусков версий. 12](#_Toc388731991)

[2 Этап проектирования (Развитие) 13](#_Toc388731992)

[3 Этап построения 13](#_Toc388731993)

[1 Студент 14](#_Toc388731994)

[1) Полностью определить и реализовать прецеденты (по варианту). 14](#_Toc388731995)

[2 Студент 17](#_Toc388731996)

[1) Полностью определить и реализовать прецеденты (по варианту). 17](#_Toc388731997)

[Тестирование и отладка 20](#_Toc388731998)

# Варианты и распределение студентов.

Работу выполняли:

Студент 1 – Тихонов И. В. вариант 92

Паттерны: модуль таблицы (бизнес-логика) и преобразователь данных

(обращение к БД), мост (gof)

Студент 2 – Лахвич Д. С. вариант 90

Паттерны: модуль таблицы (бизнес-логика) и шлюз записи данных (обращение к

БД), компоновщик (gof).

# 1. Этап анализа и планирования требований

## Исходные данные

Журнал операций: подсистемы

1) фиксация сообщений от внешних систем (сервис или утилита)

2) просмотр и поиск по журналу сообщений

3) экспорт журнала в другие форматы

## Постановка задачи

Необходимо разработать систему для работы с журналом сообщений. Система должна фиксировать сообщения от других систем и выполнять поиск по журналу сообщений. Программа может принимать сообщения например об ошибках других программ. Областью применения данной программы будут компании которые занимаются тестированием приложений, исследованиями пользовательского поведения и др. Курсовая работа выполняется двумя студентами. В рамках этого проекта будут рассмотрены 2 подсистемы

1) фиксация сообщений от внешних систем (сервис и утилита)

2) просмотр и поиск по журналу сообщений

## Спецификация основных проектных требований

Система «Журнал операций» представляет собой систему с интерфейсом для работы с сообщениями полученными от внешних программ. Интерфейс пользователя позволяет пользователю:

1. Получать список подписанных программ – клиентов
2. Получать список полученных сообщений
3. Проводить фильтрацию сообщений по программе-клиенту
4. Производить поиск сообщений по совпадению подстроки.
5. Вести поиск в БД сообщений в ручном режиме
6. Экспортировать журнал в другие форматы

Интерфейс пользователя должен отображать следующие поля:

1. Таблицу журнала сообщений
2. Таблица подписанных программ-клиентов
3. Форма поиска
4. Форма экспорта

## Описание бизнес процессов.

Моделирование прецедентов (вариантов использования) традиционно разделяют на бизнес-моделирование (диаграммы бизнес-прецедентов) и системное моделирование (диаграммы прецедентов). В бизнес-моделировании делается акцент на саму организацию, тогда как в системном моделировании основное внимание нацелено на разрабатываемую систему.

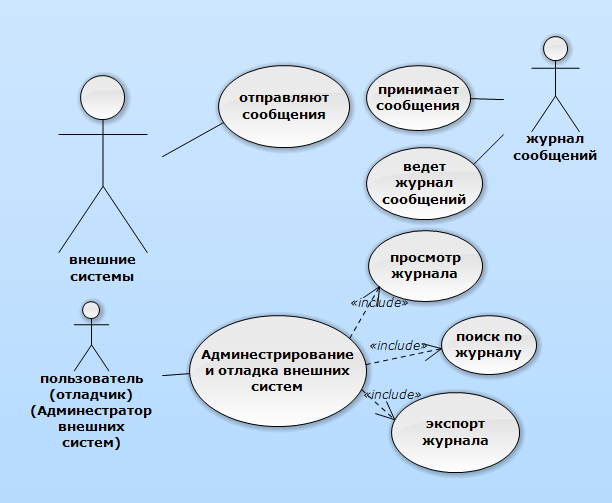


Рисунок 1. Диаграмма бизнес прецедентов.

## Модель предметной области

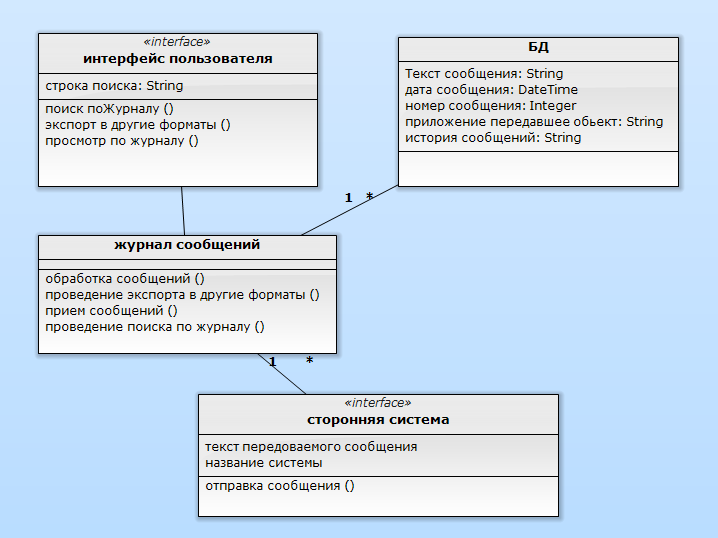


Рисунок 2. Диаграмма классов

На рисунке 2 описана модель предметной области. Она состоит из двух интерфейсов, трех сущностей и трех управляющих классов.

Интерфейсы:

* Интерфейс сторонней системы (программист или внешняя система). Имеет поля ввода сообщения, список с уровнями логгирования и кнопку запуска программы. В случае «сторонней системы» она представляется неким «черным ящиком» отправляющим готовые сообщения, которые перехватывает система.
* Интерфейс пользователя - основной интерфейс пользователей программного продукта. Содержит: окно поиска по БД и диалоговое окно экспорта, а так же журнал в виде таблицы.

Сущности:

* Сущность БД содержит данные о программе, такие как: Название, Издатель, Номер сообщения, Время получения сообщения, статус сообщения, текст сообщения, программный модуль выдавший сообщение.

Управляющие классы:

Журнал сообщений- содержит функции по приему сообщений от внешних систем, по поиску по журналу сообщений, функцию по обработки сообщений и экспорту в другие формаы.

## Выявленные актеры.

В нашей системе будет три актера.

1. Пользователь( потребитель ), который использует программу – клиент для получения отправленных сообщений и другой манипуляции с ними.
2. Сторонняя система(программа – клиент), программа, которая отправляет в журнал операций для фиксации различные сообщения.
3. Система( библиотека ),которая занимается приёмом от различных программ-клиентов, фиксацией и другими различными манипуляциями с сообщениями. А также передачей пользователю информации о сообщениях.

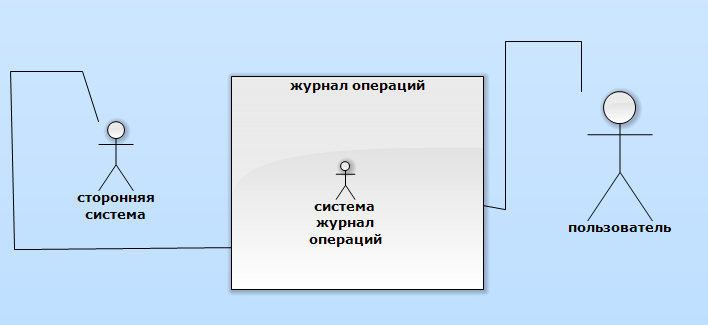


Рисунок 3. выявленные актеры журнала сообщений

## Выявленные прецеденты

* Отправка сообщения
* Получение сообщения
* Запись сообщения в БД
* Вывод сообщений
* Поиск по журналу сообщений
* Экспорт в другие форматы

## Диаграмма прецедентов

Диаграмма основных прециндентов Use Case, показана на рисунке 4.

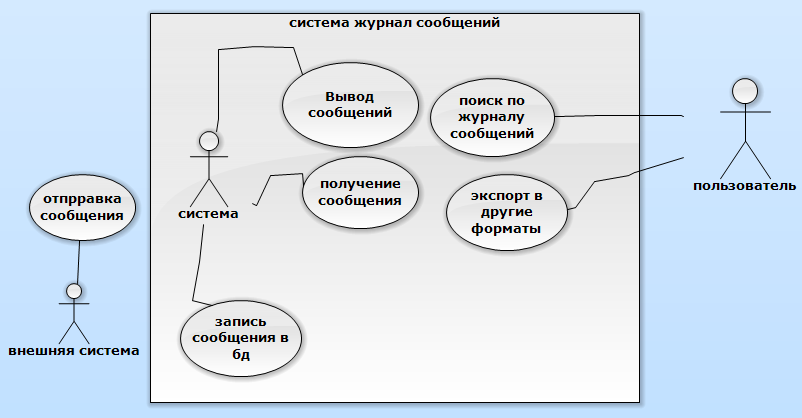


Рисунок 4. диаграмма прецедентов для системы «журнал сообщений»

## Описание прецедентов

* Отправка сообщения: сторонняя система или программист отправляет сообщения системе «журнал сообщений»
* Получение сообщения: система «журнал сообщений» принимает сообщение от сторонней системы.
* Запись сообщения в БД: принятое сообщение записывается в базу данных.
* Вывод сообщений: по запросу пользователя из БД система выводит на экран журнал сообщений.
* Поиск по журналу сообщений: позволяет пользователю вести поиск по журналу сообщений, и просматривать оставленные сообщения
* Экспорт в другие форматы: позволяет пользователю сохранять журнал сообщений в других форматах.

## Перечень критических рисков

Выделяют три категории источников риска: проектный риск, технический риск, коммерческий риск.

Для данного проекта наиболее значимый коммерческий риск, и иметься проектный риск. Технический риск заключается в сложности реализации подобного рода систем работающих под высокой нагрузкой. Поскольку на рынке уже имеются такие системы, коммерческие риски наиболее значительны. Коммерческие риски значимые для этого проекта:

1. создание продукта, не требующегося на рынке;
2. создание продукта, отстающего от требований рынка;

Поэтому следует скорее запустить приложение и собрать первые отклики пользователей, для улучшения коммерческой привлекательности приложения.

Значимые проектные риски: не уложиться в бюджет, нехватка ресурсов и персонала.

## Список приоритетов прецедентов.

* Приоритет 1. Наиболее приоритетные прецеденты: Отправка сообщений, Получение сообщений, Запись сообщения в БД.
* Приоритет 2. Средние по значимости прецеденты: Поиск по журналу сообщений, Вывод сообщений.
* Приоритет 3. Низкие по значимости прецеденты: экспорт в другие форматы. ( отправкой сообщений занимается другая система).

## Описание возможной архитектуры

Изначальный выбор архитектуры, учитывает основные технические и функциональные структурные компоненты и связи между ними. На рисунке 5 показана базовая архитектура журнала сообщений.

перечень архитектурно-значимых прецедентов?

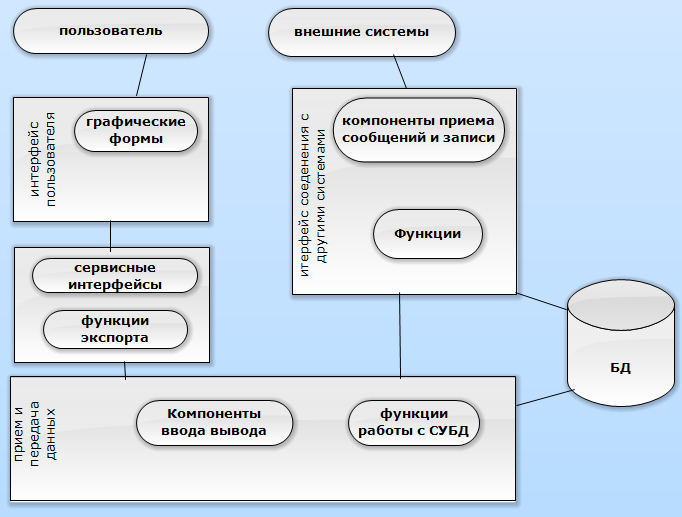


Рисунок 5. Базовая архитектура журнала сообщений.

Обобщенные механизмы проектирования:

Буем использовать каскадную модель проектирования (классический жизненный цикл), поскольку он дает план и временной график по всем этапам проекта, упорядочивает ход конструирования. Проект нам надо сдать уже готовый.

Системное ПО:

Кодирование будм производить в среде visual studio 2013, диаграммы на Software IdeasModeller, MS Project, БД Microsoft sql server, на Windows 8, Поскольку эти продукты быстрые, надежные и имеются в наличии.

Планируется использовать языки C# , SQL, приложение планируеться типа Forms Application. // добавим реально используемые компоненты.

// необходимо привести варианты, их достоинства и недостатки и обоснование выбора. ПОЗЖЕ СДЕЛАЕМ

## Начальная оценка стоимости, затрат и длительности проекта

Используется конструктивная модель стоимости (COCOMO II), этапа композиции приложения на основе объектных указателей.

Система использует 1 экран и 1 отчет. Экран используемый системой, это экран интерфейса пользователя, определяется как «простой» (количество представлений <7 и <4 таблиц данных). Отчет так же является простым так как содержит одну таблицу данных.

Процент повторного использования программных компонентов %REUSE равен нулю (новый проект). Таким образом, количество новых объектных указателей (NOP) = (Объектные указатели) х [(100 - %REUSE) /100] = 3 х (100 – 0)/100 = 3.

Скорость разработки (PROD), рассчитывается как номинальная (учитывая высокую зрелость среды разработки и низкий опыт разработчиков). PROD = 13.

Затраты, вычисляются формулой:

ЗАТРАТЫ = NOP /PROD [чел.-мес] = 3/13 = 0,23 [чел.-мес]

Далее, вычисляется стоимость проекта, учитывая среднее значение рабочего коэффициента равен $1 000 за чел.-месяц (рекомендуется брать 15000 но студенты мало получают):

СТОИМОСТЬ = ЗАТРАТЫ х РАБ\_ КОЭФ = 0,23 х $1000 = $230

Для вычисления оценки длительности разработки используется расчет характеристики масштабных факторов, показанный на таблице 2.

*Таблица 2.*

|  |  |  |
| --- | --- | --- |
| **Масштабный фактор (*Wi*)** | **Значение** | **Описание** |
| Предсказуемость PREC | 4 | низкий опыт в разработке данной программы |
| Гибкость разработки FLEX | 4 | клиент установил общие цели, и дал идеи осуществляемого процесса |
| Разрешение архитектуры /риска RESL | 3 | средний анализ рисков |
| Связность группы TEAM | 2 | уровень взаимодействия выше среднего |
| Зрелость процесса РМАТ | 3 | для оценки зрелости процесса в организации, берется номинальное значение |

**

*В =* 1,01 + 0,01 (4 + 4 + 3 + 2 + 3) = 1,17

После определения затрат и стоимости отцениваем длительность разработки:

*TDEV* = [3,0х(ЗАТРАТЫ)(0,33+0,2(B-1,01))] х SCEDPercentage/100 [мес]

SCEDPercentage — процент увеличения (уменьшения) номинального графика. Учитывая рекомендацию, полученную при анализе проектного риска, имеем SCEDPercentage = 40. Таким образом, получаем длительность:

TDEV = [3,0х(0,23)(0,33+0,2(1,17-1,01))]х 40/100 = 0.7 [мес]

## Начальный план выпусков версий.

На рисунках 6,7,8 показан начальный план выпусков версий, выполненный на MS Project.

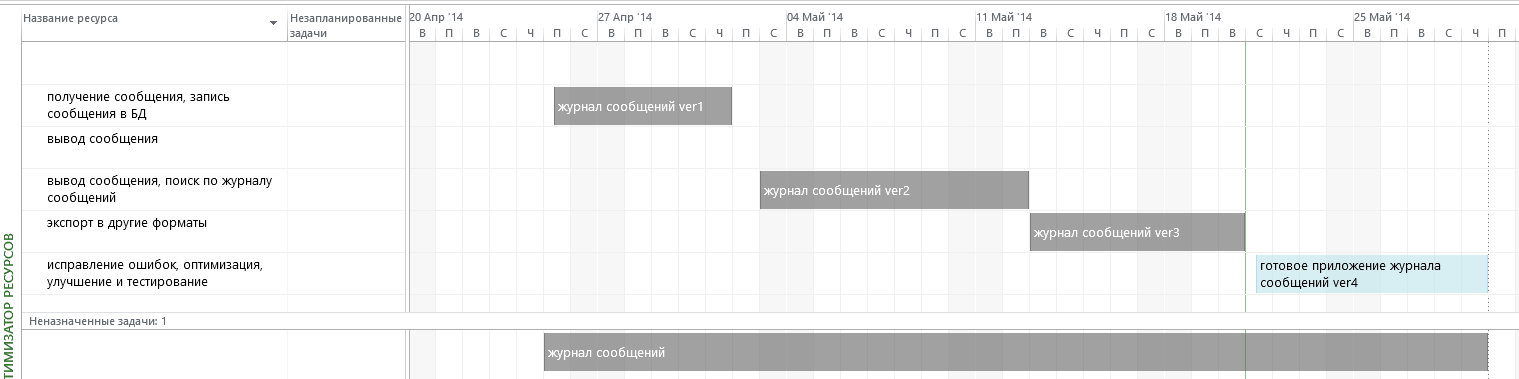


Рисунок 6. Начальный план выпусков версий.

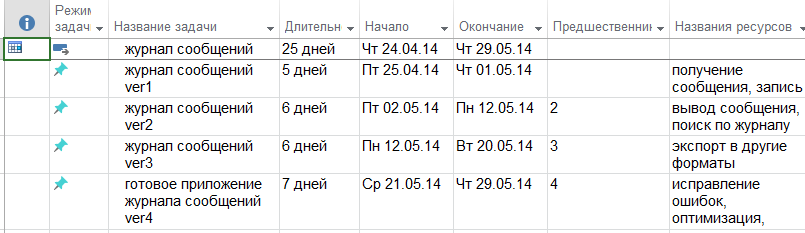


Рисунок7. Начальный план выпусков версий.

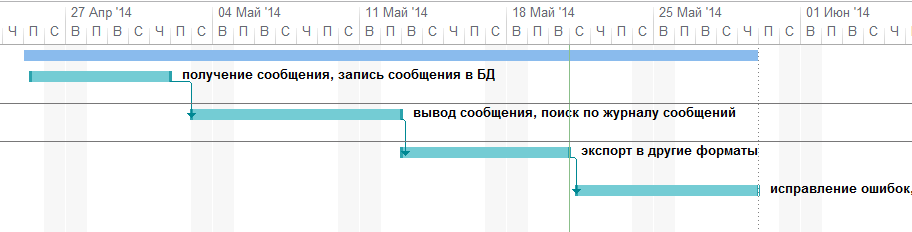


Рисунок 8. Начальный план выпусков версий.

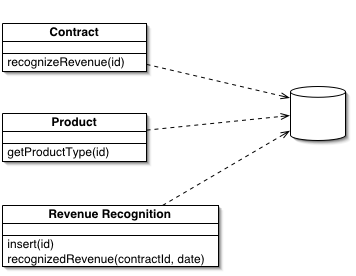
# Этап проектирования (Развитие)

2.1

# Этап построения

Паттерны:

Модуль таблицы [ Table Modyle ] это объект, охватывающий логику обработки всех записей хранимой или виртуальной таблицы базы данных.



Одна сущность обрабатывает всю бизнес-логику для всех строк таблице БД или виде.

Один из основополагающих принципов в ООП - сочетание данных и методов обработки этих данных. Традиционный объектно-ориентированный подход основан на объектах с соответствием, как, например, в паттерне Domain Model. Таким образом, если есть класс Employee, люой экземпляр этого класса соответствует конкретному работнику. Эта структура работает хорошо, потому что, имея связь, можно выполнять операции, использователь отношения, и собирать данные о работнике

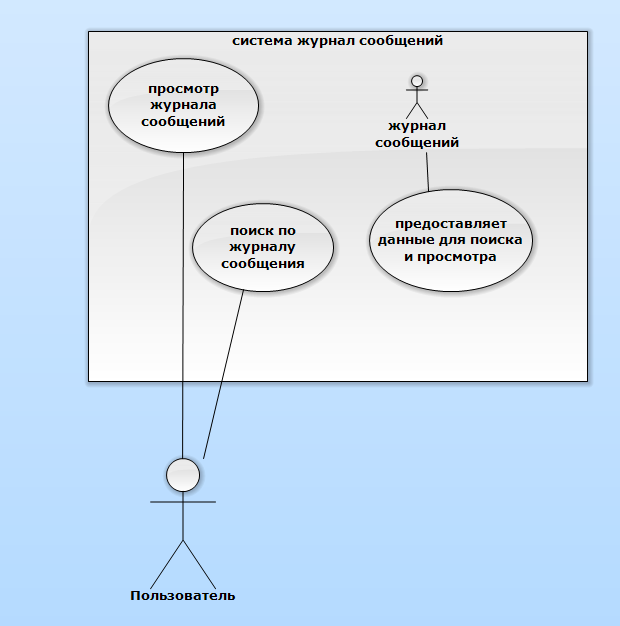
Одна из проблем в паттерне Domain Model заключается в интерфейсе к БД. Этот подход относится к БД, как к сумашедшей тётушке, запертой на чердаке, с которой никто не хочет говорить. Частенько, приходится сильно постараться, чтобы записать или считать данные из БД, преобразуя их между двумя представлениями.

Паттерн Table Module разделяет логику области определения (домена) на отдельные классы для каждой таблицы в БД и один экземпляр класса содержит различные процедуры, работающие с данными. Основное отличие от Domain Model заключается в том, что если есть несколько заказов, то Domain Model будет создавать для каждого из заказов свой объект, а Table Module будет управлять всем заказами при помощи одного объекта.

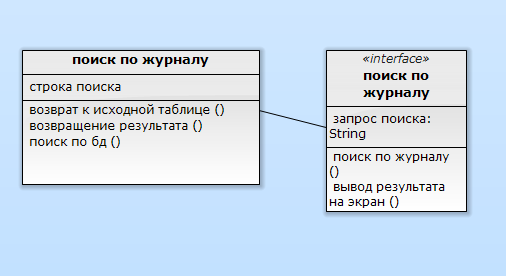
## 1 Студент

### Полностью определить и реализовать прецеденты (по варианту).

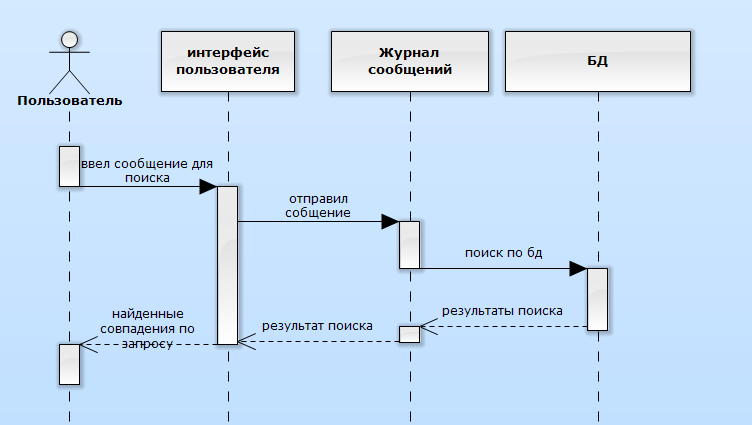
Диаграмма прецедентов для 1 студента.



1. Диаграмма(ы) классов подсистемы



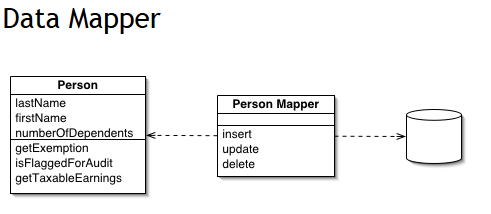
1. Полностью разработать компоненты с использованием паттернов (по вариантам).
2. Диаграммы последовательностей для основных функций программы,



1. Диаграммы классов и последовательностей для иллюстрации примененных паттернов.

модуль таблицы (описан выше)

преобразователь данных (обращение к БД)



Описание Data Mapper

Объектные и реляционные БД используют разные способы структурирования данных. Множество составляющих объектов, например коллекции и наследование, не представлены в реляционных БД. Когда проектируется объектная модель с большим количеством бизнес-логики, полезно применять такие механизмы для улучшения организации хранения данных и логики, которая работает c ними. Это приводит к различиям в организации. Так что объектная и реляционная схемы не идентичны.

Тем не менее, необходимость в обмене данными между двумя схемами не отпадает, и этот обмен становится, в свою очередь, сложным. Если же объект знает о реляционной структуре — изменения в одной из структур приведёт к проблемам в другой.

Data Mapper — это программная прослойка, разделяющая объект и БД. Его обязанность — пересылать данные между ними и изолировать их друг от друга. При использовании Data Mapper'а объекты не нуждаются в знании о существовании БД. Они не нуждаются в SQL-коде, и (естественно) в информации о структуре БД. Так как Data Mapper - это разновидность паттерна Mapper, сам объект-Mapper неизвестен объекту.

мост (gof)

Когда абстракция и реализация разделены, они могут изменяться независимо. Другими словами, при реализации через паттерн мост, изменение структуры интерфейса не мешает изменению структуры реализации.

1. Оценка качества проекта.

**Итерация 1**

Классы:

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| Form1 | private lib.WinHistory history; | public Form1() |
| private List<lib.Models.Message> messages; | private void searchButton\_Click(object sender, EventArgs e) |
| private List<lib.Models.ClientInfo> clients; | private void exportButton\_Click(object sender, EventArgs e) |
|  |  |
|  |  |
|  |  |
|  |  |

**Итерация 2**

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| class Form1 | lib.WinHistory history | public Form1() |
| List<lib.Models.Message> messages | private void searchButton\_Click(object sender, EventArgs e) |
| List<lib.Models.ClientInfo> clients | private void exportButton\_Click(object sender, EventArgs e) |
|  | private void ResfreshMessages() |
|  | private void ClientView\_CellEndEdit(object sender, DataGridViewCellEventArgs e) |
| private void containsQuery\_TextChanged(object sender, EventArgs e) |
| private void minLevelQuery\_TextChanged(object sender, EventArgs e) |
| private void button1\_Click(object sender, EventArgs e) |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| class Program |  | static void Main() |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| interface ILogExport | string FilterName | void ExportToFile(string path, IEnumerable<Message> messages); |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| class LogExportToTXT | string ILogExport.FilterName | void ILogExport.ExportToFile(string path, IEnumerable<Message> messages) |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| class LogExportToXML | string ILogExport.FilterName | void ILogExport.ExportToFile(string path, IEnumerable<Message> messages) |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| class LogExporter | private List<ILogExport> typesOfExport | public LogExporter(List<ILogExport> types) |
| public void Export(string path, int index, IEnumerable<Message> messages) |
| public string Filter |

Оценка качества проекта:

Метрика 1: Взвешенные методы на класс WMC (WeightedMethodsPerClass)

Применим упрощенную версию метрик –WMC будет равняться количеству методов в  классе. Подсчитываются только  методы  текущего  класса.  Унаследованные  методы  игнорируются. Метрика  WMC  дает  относительную  меру  сложности  класса. Считаем,  что  все методы класса имеют  одинаковую  сложность.

WNC class Form1=8

WNC class Program=1

WNC interface ILogExport=1

WNC class LogExportToTXT=1

WNC class LogExportToXML=1

WNC class LogExporter =3

общий WNC=2.5

Метрика 2: Количество детей NOC (Number of children)

Подклассы, которые непосредственно подчинены суперклассу, называются его детьми. Значение NOC равно количеству детей, то есть количеству непосредственных наследников класса в иерархии классов

NOC interface ILogExport =2

NOC=0

Метрика 3: Сцепление между классами объектов СВО (Coupling between object classes)

СВО — это количество сотрудничеств, предусмотренных для класса, то есть количество классов, с которыми он соединен. Соединение означает, что методы данного класса используют методы или экземплярные переменные другого класса. Данная метрика характеризует статическую составляющую внешних связей классов. С ростом СВО многократность использования класса, вероятно, уменьшается. Очевидно, что чем больше независимость класса, тем легче его повторно использовать в другом приложении. Высокое значение СВО усложняет модификацию и тестирование, которое следует за выполнением модификации. Понятно, что, чем больше количество сцеплений, тем выше чувствительность всего проекта к изменениям в отдельных его частях. Минимизация межобъектных сцеплений улучшает модульность и содействует инкапсуляции проекта. СВО для каждого класса должно иметь разумно низкое значение. Это согласуется с рекомендациями по уменьшению сцепления стандартного программного обеспечения.

СВО class Form1=5

СВО class Program=1

СВО interface ILogExport=1

СВО class LogExportToTXT=1

СВО class LogExportToXML=1

СВО class LogExporter =1

СВО=1,6

**Метрика 4: Отклик для класса RFC (Response For a Class)**

RFC — это количество методов класса плюс количество методов других классов, вызываемых из данного класса.

Метрика RFC является мерой потенциального взаимодействия данного класса с другими классами, позволяет судить о динамике поведения соответствующего объекта в системе. Данная метрика характеризует динамическую составляющую внешних связей классов.

Если в ответ на сообщение может быть вызвано большое количество методов, то усложняются тестирование и отладка класса, так как от разработчика тестов требуется больший уровень понимания класса, растет длина тестовой последовательности.

С ростом RFC увеличивается сложность класса. Наихудшая величина отклика может использоваться при определении времени тестирования

RFC class Form1=8+0=8

RFC class Program=1+3=4

RFC interface ILogExport=1+0=1

RFC class LogExportToTXT=0+1=1

RFC class LogExportToXML=0+1=1

RFC class LogExporter =3+0=3

RFC =3

Метрика 5: Недостаток связности в методах LСOM (Lack of Cohesion in Methods)

Каждый метод внутри класса обращается к одному или нескольким свойствам (экземплярным переменным). Метрика LCOM показывает, насколько методы не связаны друг с другом через свойства (переменные). Если все методы обращаются к одинаковым свойствам, то LCOM = 0. Можно определить метрику по-другому: LCOM — это количество пар методов, не связанных по свойствам класса, минус количество пар методов, имеющих такую связь.

Метрика 6: Размер класса CS (Class Size)

Общий размер класса определяется с помощью следующих измерений:

- общее количество операций (вместе с приватными и наследуемыми экземплярными операциями), которые инкапсулируются внутри класса;

- количество свойств (вместе с приватными и наследуемыми экземплярными свойствами), которые инкапсулируются классом.

Могут вычисляться средние количества свойств и операций класса. Чем меньше среднее значение размера, тем больше вероятность повторного использования класса.

Рекомендуемое значение CS ≤ 20 методов.

Метрика 7: Количество операций, переопределяемых подклассом, NOO

(Number of Operations Overridden by a Subclass)

Переопределением называют случай, когда подкласс замещает операцию, унаследованную от суперкласса, своей собственной версией.

Большие значения NOO обычно указывают на проблемы проектирования. Ясно, что подкласс должен расширять операции суперкласса. Расширение проявляется в виде новых имен операций. Если же NOО велико, то разработчик нарушает абстракцию суперкласса. Это ослабляет иерархию классов, усложняет тестирование и модификацию программного обеспечения.

Рекомендуемое значение NOO ≤ 3 методов.

Метрика 8: Количество операций, добавленных подклассом, NOA

(Number of Operations Added by a Subclass)

Подклассы специализируются добавлением приватных операций и свойств. С ростом NOA подкласс удаляется от абстракции суперкласса. Обычно при увеличении высоты иерархии классов (увеличении DIT) должно уменьшаться значение NOA на нижних уровнях иерархии. Для рекомендуемых значений CS = 20 и DIT = 6 рекомендуемое значение NOA ≤ 4 методов (для класса-листа).

NOA=0

Метрика 9: Индекс специализации SI (Specialization Index)

Обеспечивает грубую оценку степени специализации каждого подкласса. Специализация достигается добавлением, удалением или переопределением операций:

SI = (NOO \* уровень) /Mобщ,

где уровень — номер уровня в иерархии, на котором находится подкласс, Мобщ — общее количество методов класса.

Чем выше значение SI, тем больше вероятность того, что в иерархии классов есть классы, нарушающие абстракцию суперкласса.

Рекомендуемое значение SI ≤ 0,15.

Метрика 10: Средний размер операции OSAVG (Average Operation Size)

В качестве индикатора размера может использоваться количество строк программы.

Метрика 11: Среднее количество параметров на операцию NPAVG

(AverageNumberofParametersperoperation)

Чем больше параметров у операции, тем сложнее сотрудничество между объектами. Поэтому значение NPAVG должно быть как можно меньшим.

Рекомендуемое значение NPAVG = 0,7.

Метрика 12: Высота дерева наследования DIT (Depth of Inheritance Tree)

DIT определяется как максимальная длина пути от листа до корня дерева наследования классов.

DITАИС =

Метрика 13: Суммарное количество методов, определенных во всех классах системыNOM (Numberofmethods)

Метрика равна количеству методов во всех классах системы.

NOMАИС = 8

Метрика 14: Общее количество классов в системе – NC (Numberofclasses)

Метрика равна общему количеству классов в системе.

NCАИС = 6

Метрика 15: Общее число строк кода – LOCsum (LinesOfCode)

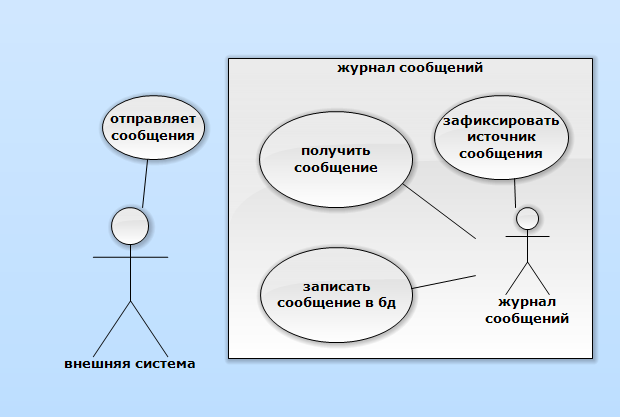
Метрика равна общему числу строк кода системы.

LOCsumАИС =

## 2 Студент

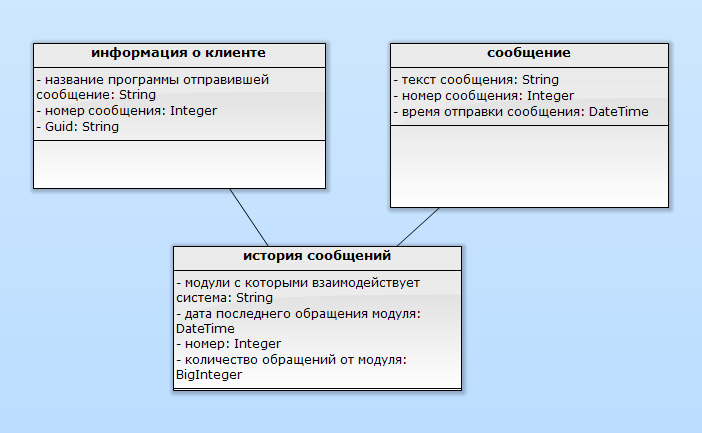
### Полностью определить и реализовать прецеденты (по варианту).

Диаграмма прецедентов для 2 студента.

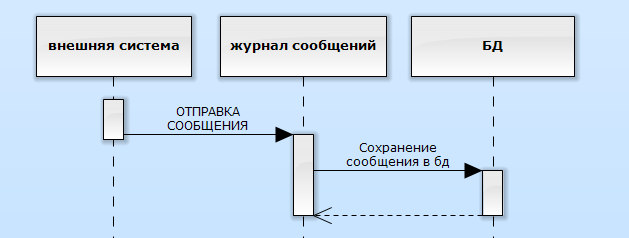


1. Диаграмма(ы) классов подсистемы,

Описание БД системы журнала сообщений



1. Полностью разработать компоненты с использованием паттернов (по вариантам).
2. Диаграммы последовательностей для основных функций программы

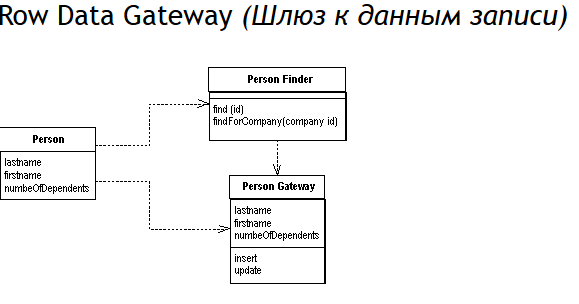


1. Диаграммы классов и последовательностей для иллюстрации примененных паттернов.

модуль таблицы (описано выше)

шлюз записи данных (обращение к БД)

Описание Row Data Gateway



Объект выступает в роли шлюза к отдельной записи в источнике данных. Один экземпляр на одну запись.

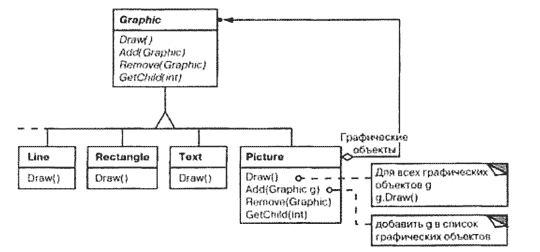
Встраивание кода доступа к БД в объекты, хранящиеся в памяти, может привести в некоторым неудобствам. Во-первых, если ваши объекты содержат бизнес-логику, добавление взаимодействия с БД увеличит сложность. Вскоре тестирование станет неудобным, если объекты, хранящиеся в памяти, завязаны на БД. Тесты станут медленнее из-за доступа к БД.

Объект шлюза к записи представляется в точности, как запись в БД, но при этом даёт возможность доступа посредством штатных средств языка программирования. Все подробности доступа к БД скрыты за этим интерфейсом.

Пример: объект шлюза PersonGateway, данные о записи из таблицы person и методы insert и update, которые позволяют прозрачно работать с записью.

компоновщик (gof).

Компоновщик - паттерн, структурирующий объекты таким образом, что появляется возможность унифицировано, одинаково обращаться с каждым из них.



Ключом к паттерну компоновщик является абстрактный класс, который является одновременно и примитивом, и контейнером. В графической системе этот класс может называться Graphic. В нем объявлены операции, специфичные для каждого вида графического объекта (такие как Draw) и общие для всех составных объектов, например операции для доступа и управления потомками. Подклассы Line, Rectangle и Text определяют примитивные графические объекты. В них операция Draw реализована соответственно для рисования прямых, прямоугольников и текста. Поскольку у примитивных объектов нет потомков, то ни один из этих подклассов не реализует операции, относящиеся к управлению потомками(Add, Remove, GetChild).

Класс Picture агрегирует, состоит из других более примитивных объектов Graphic. Реализованная в нем операция Draw вызывает одноименную функцию отрисовки для каждого потомка, а операции для работы с потомками уже не пусты. Поскольку интерфейс класса Picture соответствует интерфейсу Graphic, то в состав объекта Picture могут входить и другие такие же объекты.

1. Провести тестирование и отладку.

**Итерация1**

Классы:

|  |  |  |
| --- | --- | --- |
| Класс | поля | методы |
| WinHistory |  | private WinHistory() |
| WinHistory(Guid guid, string name) |
| public static WinHistory Login(Guid guid, string name) |
| public void Send(string message, int level) |
| public void Send(string message) |
| public IEnumerable<Message> Receive(int level) |
| public IEnumerable<Message> Receive() |
| public IEnumerable<Message> Receive(Guid guid) |
| public IEnumerable<Message> Receive(Guid guid, int level) |
| public IEnumerable<Message> Receive(string contains) |
| public IEnumerable<ClientInfo> ReceiveClients() |
| public ClientInfo ReciveClientInfo(Guid guid) |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| ClientInfo | public int ClientInfoId | get; |
| public string Guid | set; |
| public string Name |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| Message | public int MessageI | get; |
| public ClientInfo ClientInfo |
| public string Text | set; |
| public DateTime Stamp |
| public int Level |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| WinHistoryDbContext | public DbSet<Message> Messages | public WinHistoryDbContext() |
| public DbSet<ClientInfo> Clients | get |
|  | set |

Тестирование и отладка