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. 4](#_Toc389138462)

[1. Этап анализа и планирования требований 4](#_Toc389138463)

[Исходные данные 4](#_Toc389138464)

[Постановка задачи 4](#_Toc389138465)

[Спецификация основных проектных требований 4](#_Toc389138466)

[Описание бизнес процессов. 5](#_Toc389138467)

[Модель предметной области 8](#_Toc389138468)

[Выявленные актеры. 10](#_Toc389138469)

[Выявленные прецеденты 11](#_Toc389138470)

[Диаграмма прецедентов 11](#_Toc389138471)

[Описание прецедентов 12](#_Toc389138472)

[Перечень критических рисков 12](#_Toc389138473)

[Список приоритетов прецедентов. 13](#_Toc389138474)

[Описание возможной архитектуры 13](#_Toc389138475)

[Начальная оценка стоимости, затрат и длительности проекта 15](#_Toc389138476)

[Начальный план выпусков версий. 16](#_Toc389138477)

[Этап проектирования (Развитие) 17](#_Toc389138478)

[Расширенные описания прецедентов 17](#_Toc389138479)

[Диаграммы деятельности основных прецедентов 19](#_Toc389138480)

[Структурированная диаграмма прецедентов 21](#_Toc389138481)

[Прототип пользовательского интерфейса. 23](#_Toc389138482)

[классы анализа. 23](#_Toc389138483)

[Диаграммы классов анализа 26](#_Toc389138484)

[Уточненная оценка стоимости на основе функциональных указателей. 26](#_Toc389138485)

[Этап построения 29](#_Toc389138486)

[1 Студент 30](#_Toc389138487)

[1) Полностью определить и реализовать прецеденты (по варианту). 30](#_Toc389138488)

[2 Студент 44](#_Toc389138489)

[1) Полностью определить и реализовать прецеденты (по варианту). 44](#_Toc389138490)

[**Тестирование и отладка** 61](#_Toc389138491)

[структурное 61](#_Toc389138492)

[Функциональное 61](#_Toc389138493)

[Нагрузочное 62](#_Toc389138494)

[Этап внедрения 62](#_Toc389138495)

[Системные требования 62](#_Toc389138496)

[Установка 62](#_Toc389138497)

[Документация 62](#_Toc389138498)

[Требования по внедрению 62](#_Toc389138499)

[Приложение А 63](#_Toc389138500)

[1 Паттерн мост 63](#_Toc389138501)

[1.1 Основной код реализации 63](#_Toc389138502)

[1.2 Код использования 65](#_Toc389138503)

[2 Паттерн «Компоновщик» 66](#_Toc389138504)

[2.1 Основной код реализации 66](#_Toc389138505)

[3 Паттерн Модуль таблицы 68](#_Toc389138506)

[4 Паттерн Data Mapper 68](#_Toc389138507)

[5 Паттерн «Шлюз к записи данных» 68](#_Toc389138508)

[Литература 69](#_Toc389138509)

# 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.

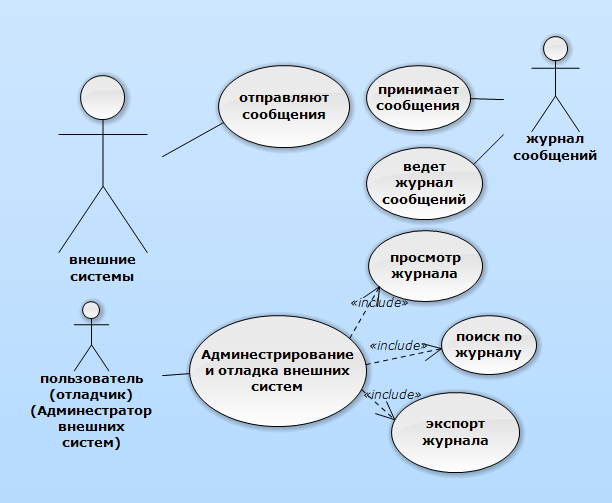


Рисунок 1. Диаграмма бизнес прецедентов.

## Модель предметной области

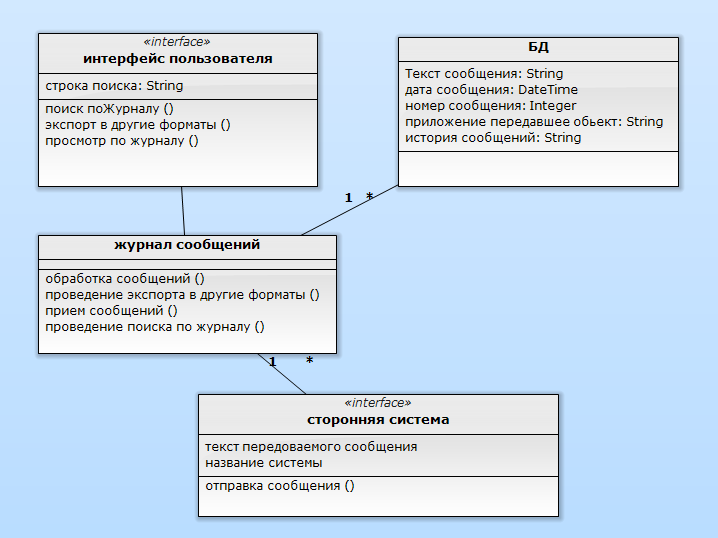


Рисунок 2. Диаграмма классов

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.

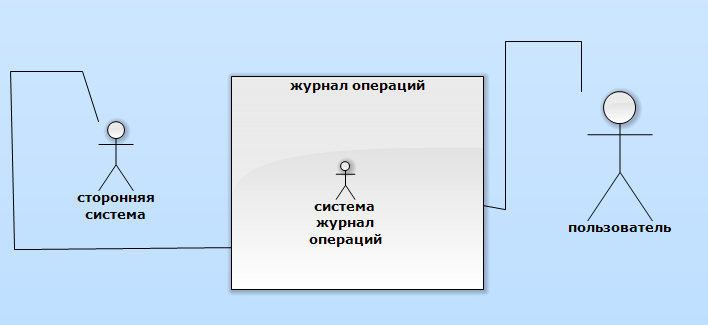


Рисунок 3. выявленные актеры журнала сообщений

## 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.

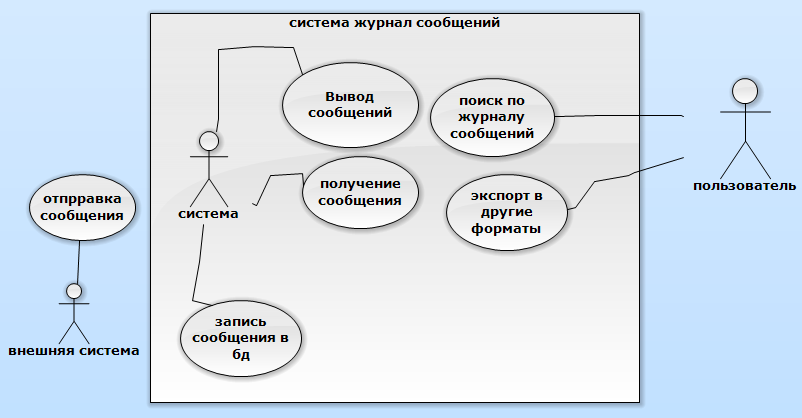


Рисунок 4. диаграмма прецедентов для системы «журнал сообщений»

## 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.

Архитектура - это***набор значимых решений***по поводу организации системы программного обеспечения, набор ***структурных элементов***и их интерфейсов, при помощи которых компонуется система, вместе с их***поведением***, определяемым во взаимодействии между этими элементами,***компоновка***элементов в постепенно укрупняющиеся подсистемы, а также***стиль архитектуры,***который направляет эту организацию -- элементы и их интерфейсы, взаимодействия и компоновку.

Архитектура программного обеспечения системы или набора систем состоит из всех важных проектных решений по поводу структур программы и взаимодействий между этими структурами, которые составляют системы. Проектные решения обеспечивают желаемый набор свойств, которые должна поддерживать система, чтобы быть успешной. Проектные решения предоставляют концептуальную основу для разработки системы, ее поддержки и обслуживания.

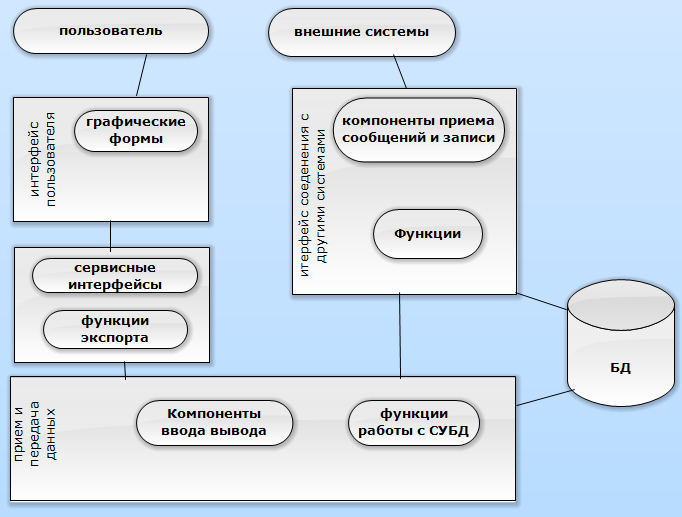


Рисунок 5. Базовая архитектура журнала сообщений.

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:

В качестве языка программирования был выбран C#,в качестве IDE решено использовать Visual studio 2013 , диаграммы на Software IdeasModeller, MS Project, БД Microsoft Sql Server Compact, на Windows 8 .NET Framework 4.5. Также была использована библиотека EF6.1 . Данный стек технологий был выбран в главной причине по тому, что он позволяет организовать максимально быстро процесс разработки под ОС Windows. В качестве альтернативы были рассмотрены платформы Qt + SQLite и Java + SqlLite. Первая была отвергнута по причине высокого порога вхождения для начала разработки, вторая по причине сложности создания системных утилит на Java. Git в качестве системы контроля версий, как стандарт де факто с расположением репозитория по адресу htts://github.com/ReiReiRei/winhistory.git.

## 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.

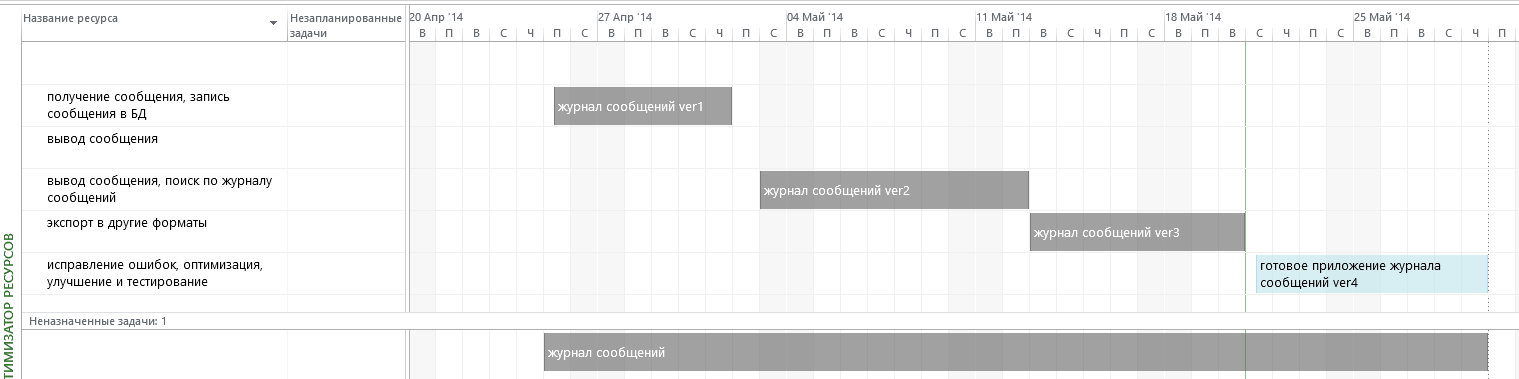


Рисунок 6. Начальный план выпусков версий.

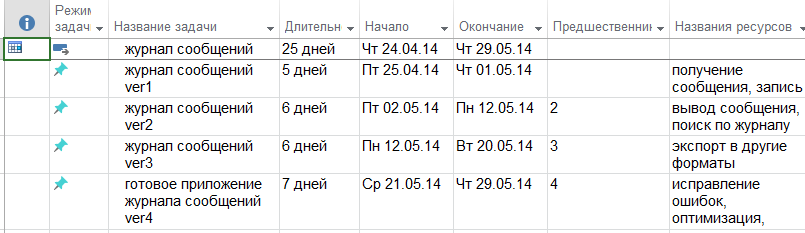


Рисунок7. Начальный план выпусков версий.

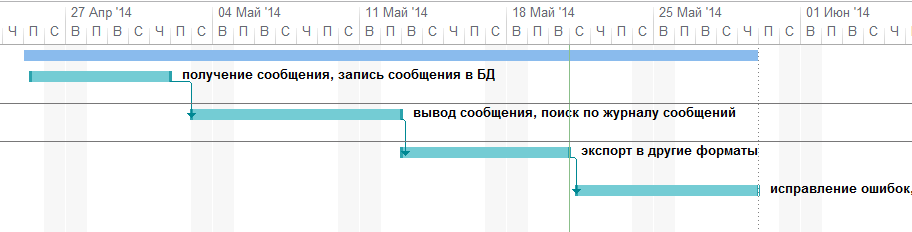


Рисунок 8. Начальный план выпусков версий.

# Этап проектирования (Развитие)

## Расширенные описания прецедентов

**Прецедент «получение сообщения»**

Краткое описание: система получает сообщение

Главные актеры: Внешняя система, система журнала сообщений.

Предусловие:

1. внешняя система отправила сообщение
2. журнал сообщений включен и настроен на прием сообщения от этой системы.

Основной поток.

1. Прецедент начинается в момент отправки сообщения.
2. Система принимает сообщение
   1. Система считывает название устройства отправляющего сообщение
   2. Система записывает сообщение и данные о нем в БД
   3. Система фиксирует номер сообщения
3. Система обновляет таблицу и ждет новых сообщений.

Постусловия: система готова к приему новых сообщений

Альтернативные потоки: система принимают сообщение от другой системы.

**Прецедент «поиск по ЖС»**

Краткое описание: пользователь ищет сообщение в журнале

Главные актеры: система журнала сообщений, пользователь

Предусловие: система включена, и имеет сообщения для поиска

Основной поток:

1. Прецедент начинается в момент входа в систему
2. Пользователь набирает фразу искомого сообщения в диалоговом окне.
   1. Система выполняет поиск по БД и выводит все совпадения по запросу
3. Иначе: пользователь ищет сообщение в интерфейсе БД
   1. . Пользователь перелистывает таблицу в поиске исходного сообщения
   2. Пользователь находит или не находит сообщение
4. Пользователь ищет новое сообщение
5. Пользователь выходит

Постусловия: система обновляется или выключается

Альтернативные потоки: выход

**Прецедент «экспорт в другие форматы»**

Краткое описание: пользователь экспортирует журнал в другие форматы

Главные актеры: система журнала сообщений, пользователь

Предусловие: система включена, и имеет сообщения для экспорта

Основной поток:

1. Прецедент начинается в момент входа в систему
2. Пользователь проводит экспорт в другие форматы.
   1. Пользователь выбирает в какой формат ему экспортировать
   2. Пользователь экспортирует в выбранный формат.

Постусловия: переход в главное меню

Альтернативные потоки: экспорт в другой формат

## Диаграммы деятельности основных прецедентов

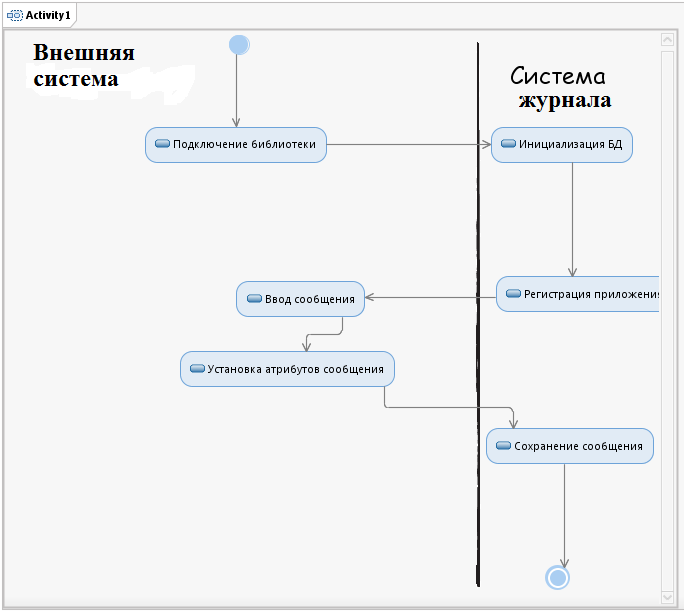


Рисунок 9 прием сообщения

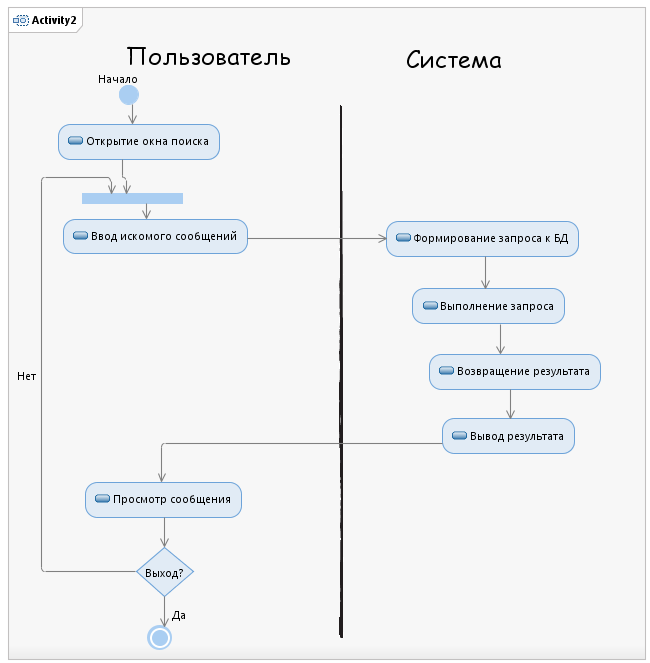


Рисунок 10. Поиск по журналу сообщений

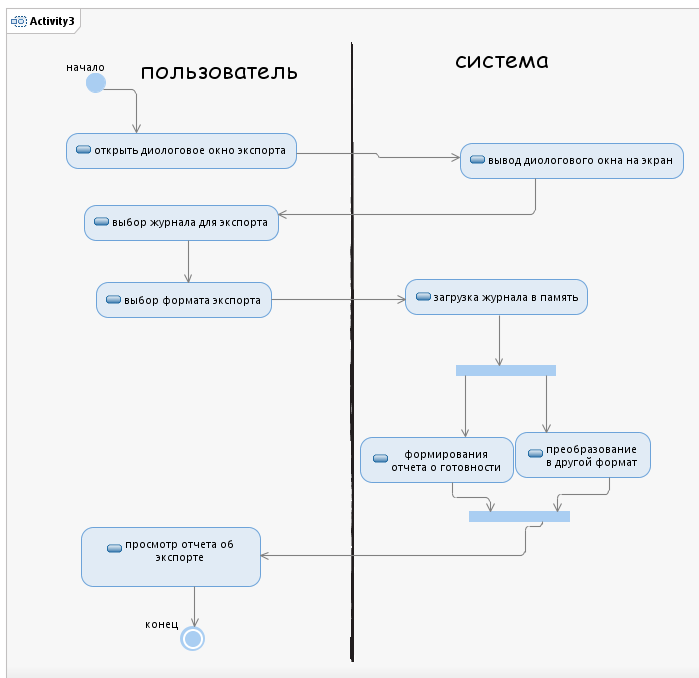


Рисунок 11. Экспорт в другие форматы.

## Структурированная диаграмма прецедентов

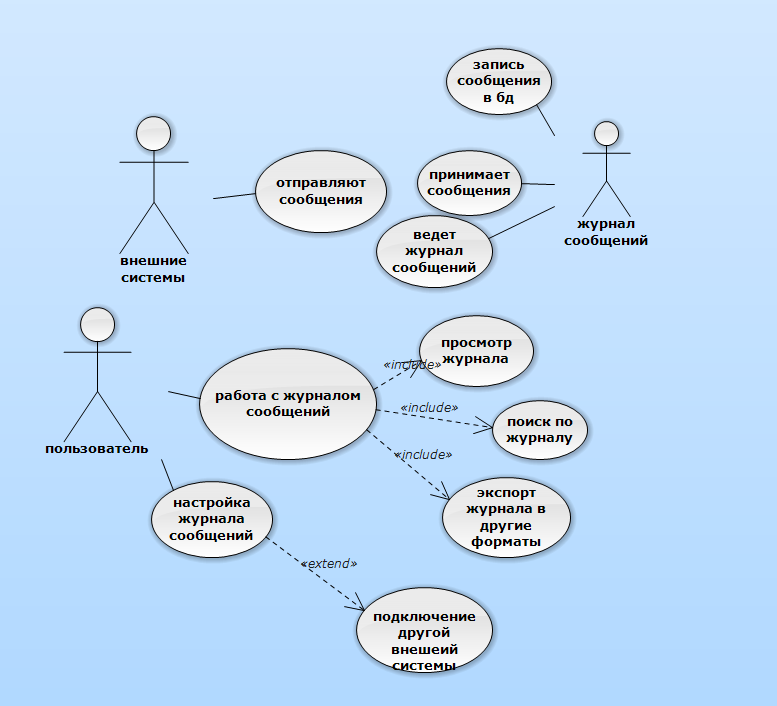


Рисунок 13. Структурированная диаграмма прецедентов.

## Прототип пользовательского интерфейса.

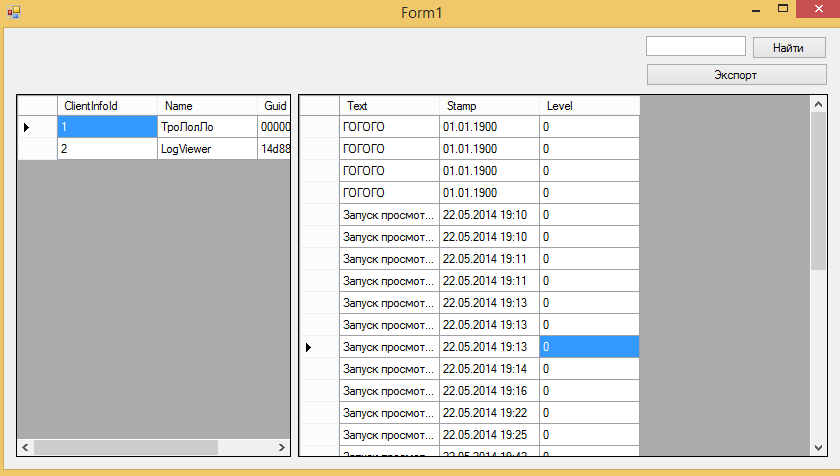
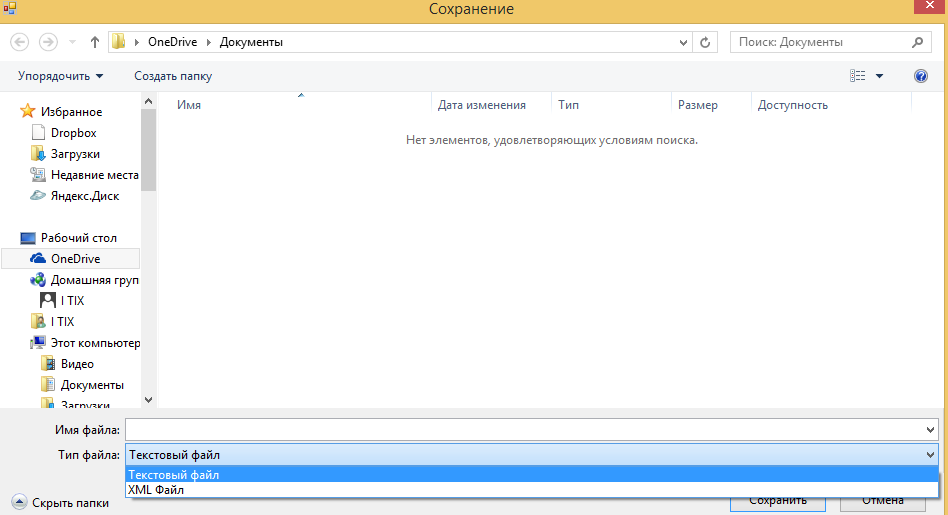


Рисунок 13. Прототип пользовательского интерфейса

рисунок 14. Интерфейс экспорта

## классы анализа.

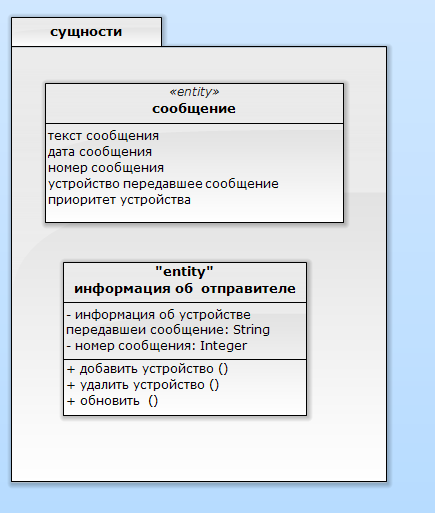


Рисунок 15.Сушьности

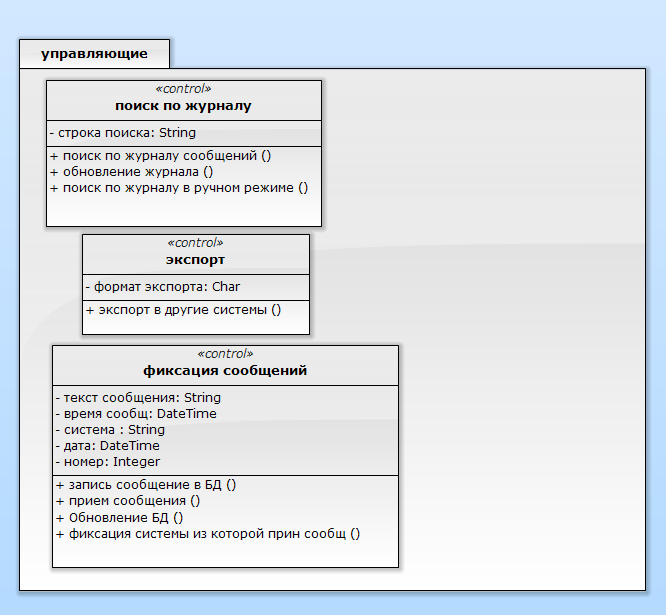


Рисунок 16.

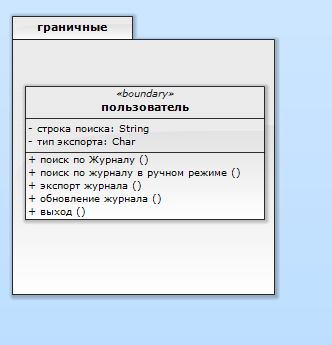


Рисунок 17. Граничные классы

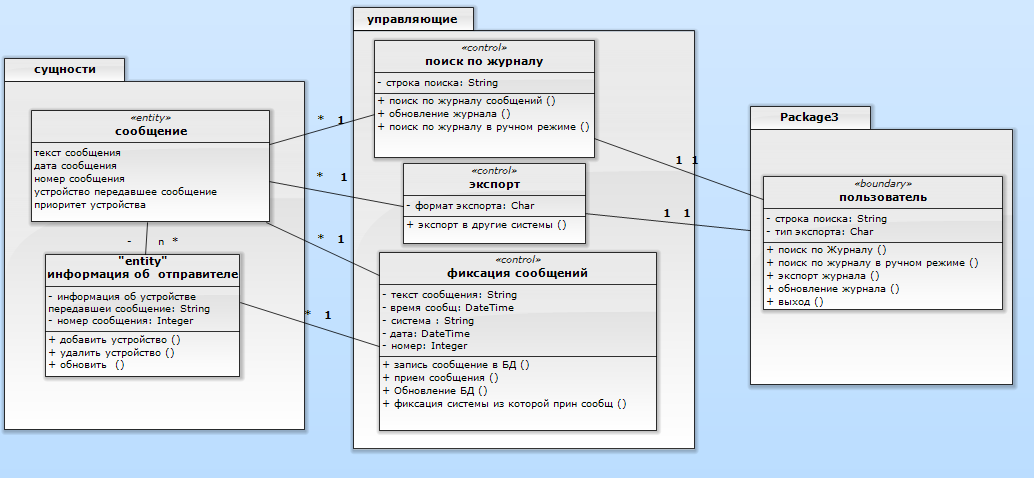


Рисунок 18.диограмма классов анализа

## Диаграммы классов анализа

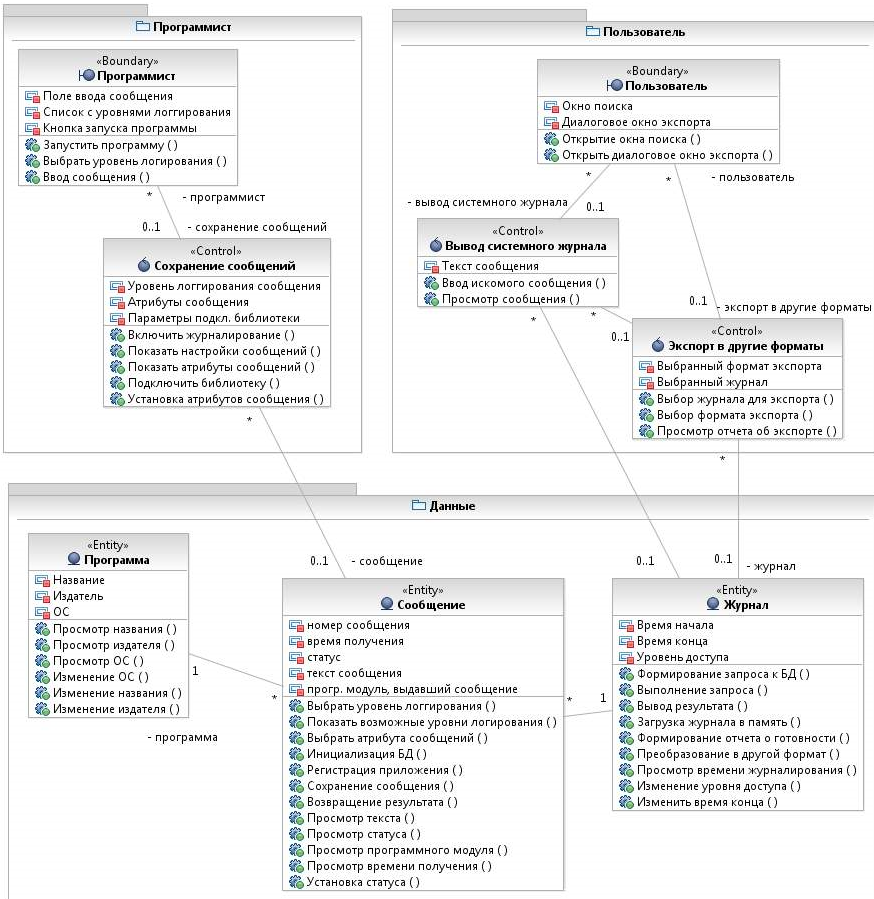
****

Рисунок 20. Диаграмма классов анализа

## Уточненная оценка стоимости на основе функциональных указателей.

На основе функциональных указателей.

Используется 5 информационных характеристик.

1. *Количество внешних вводов*. Подсчитываются все вводы пользователя, по которым поступают разные прикладные данные. Вводы должны быть отделены от запросов, которые подсчитываются отдельно.

2. *Количество внешних выводов*. Подсчитываются все выводы, по которым к пользователю поступают результаты, вычисленные программным приложением. В этом контексте выводы означают отчеты, экраны, распечатки, сообщения об ошибках. Индивидуальные единицы данных внутри отчета отдельно не подсчитываются.

3. *Количество внешних запросов*. Под запросом понимается диалоговый ввод, который приводит к немедленному программному ответу в форме диалогового вывода. При этом диалоговый ввод в приложении не сохраняется, а диалоговый вывод не требует выполнения вычислений. Подсчитываются все запросы — каждый учитывается отдельно.

4. *Количество внутренних логических файлов*. Подсчитываются все логические файлы (то есть логические группы данных, которые могут быть частью базы данных или отдельным файлом).

5. *Количество внешних интерфейсных файлов.* Подсчитываются все логические файлы из других приложений, на которые ссылается данное приложение.

Оценка информационных характеристик проекта

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя характеристики | Лучший | Вероятный | Худший | Ожидаемый | Сложность | Количество |
| Внешние вводы | 4 | 5 | 6 | 5 | 3 | 15 |
| Внешние выводы | 3 | 4 | 5 | 4 | 5 | 20 |
| Внешние запросы | 2 | 3 | 4 | 3 | 4 | 12 |
| Внутренние логические файлы | 1 | 2 | 3 | 2 | 7 | 14 |
| Внешние интерфейсные файлы | 1 | 2 | 3 | 2 | 5 | 10 |
| Общее количество |  |  |  |  |  | 72 |

Оценка системных параметров проекта

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Системный параметр** | **Описание** | **Оценка** |
| 1 | Передачи данных | Сколько средств связи требуется для передачи или обмена информацией с приложением или системой? | 5 |
| 2 | Распределенная обработка данных | Как обрабатываются распределенные данные и функции обработки? | 4 |
| 3 | Производительность | Нуждается ли пользователь в фиксации времени ответа или производительности?. | 3 |
| 4 | Распространенность используемой конфигурации | Насколько распространена текущая аппаратная платформа, на которой будет выполняться приложение? | 3 |
| 5 | Скорость транзакций | Как часто выполняются транзакции? (каждый день, каждую неделю, каждый месяц) | 5 |
| 6 | Оперативный ввод данных | Какой процент информации надо вводить в режиме онлайн? | 4 |
| 7 | Эффективность работы конечного пользователя | Приложение проектировалось для обеспечения эффективной работы конечного пользователя? | 5 |
| 8 | Оперативное обновление | Как много внутренних файлов обновляется в онлайновой транзакции? | 2 |
| 9 | Сложность обработки | Выполняет ли приложение интенсивную логическую или математическую обработку? | 0 |
| 10 | Повторная используемость | Приложение разрабатывалось для удовлетворения требований одного или многих пользователей? | 1 |
| 11 | Легкость инсталляции | Насколько трудны преобразование и инсталляция приложения? | 1 |
| 12 | Легкость эксплуатации | Насколько эффективны и/или автоматизированы процедуры запуска, резервирования и восстановления? | 2 |
| 13 | Разнообразные условия размещения | Была ли спроектирована, разработана и поддержана возможность инсталляции приложения в разных местах для различных организаций? | 4 |
| 14 | Простота изменений | Была ли спроектирована, разработана и поддержана в приложении простота изменений? | 4 |
| **Всего** |  |  | **43** |

Используя таблицы получаем

FP = Общее количество х (0,65+ 0,01 x)=72 х (0,65+0,01 х 43)=77.76

Затраты = FP / Производительность=77.76/13=5.9[чел.-мес]

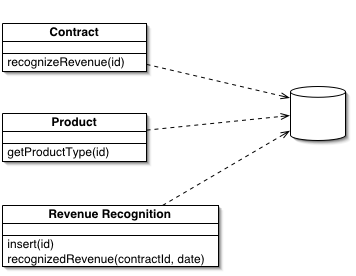
Стоимость = Затраты х $1000=$ 5900

Длительность = [3,0 х (ЗАТРАТЫ)(0,33+0,2(B-1,01))]=3(5.9)0,16 = 5.7[мес].

# 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

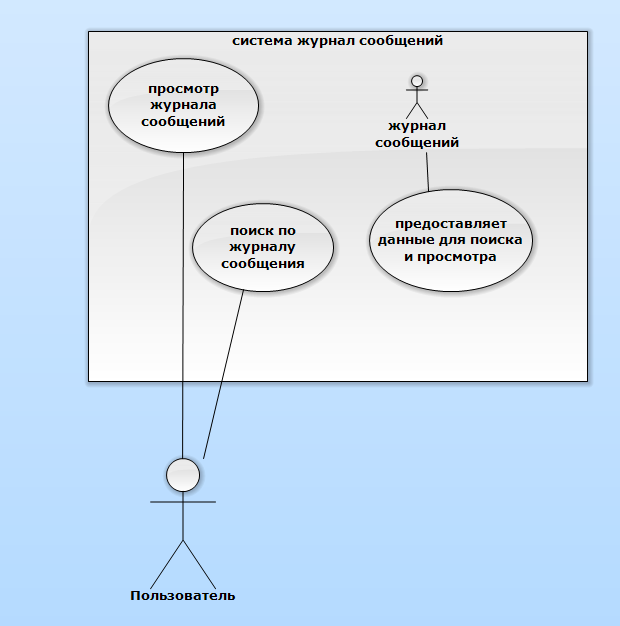
Одна из проблем в паттерне Domain Model заключается в интерфейсе к БД. Этот подход относится к БД, как к сумашедшей тётушке, запертой на чердаке, с которой никто не хочет говорить. Частенько, приходится сильно постараться, чтобы записать или считать данные из БД, преобразуя их между двумя представлениями.

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.

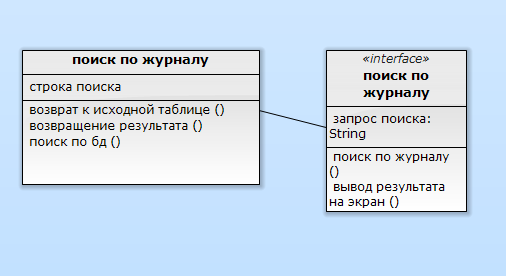
## 1 Студент

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

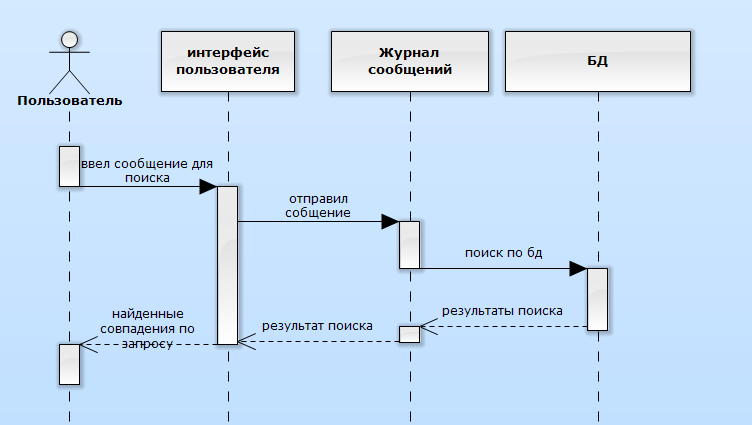
Диаграмма прецедентов для 1 студента.



1. Диаграмма(ы) классов подсистемы



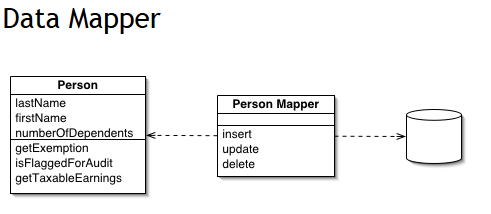
1. Полностью разработать компоненты с использованием паттернов (по вариантам).
2. Диаграммы последовательностей для основных функций программы,



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 ResfreshMessages() |
| private void containsQuery\_TextChanged(object sender, EventArgs e) |
| private void minLevelQuery\_TextChanged(object sender, EventArgs e) |
| private void button1\_Click(object sender, EventArgs e) |

**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=6

общий WNC=6

**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 =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=3

СВО=3

**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=6+5=11

RFC=11

**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.

LCOM class Form1=30-6=24

LCOM = 24

**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.

CS class Form1=6+2=8

CS =8

**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.

NOO=0

**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.

SI = 0

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

В качестве индикатора размера может использоваться количество строк программы.

OSAVG class Form1=40/6=6

OSAVG=6.66

**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.

NPAVG class Form1=3/6=0.5

NPAVG =0.5

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

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

DIT = 0

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

Метрика равна количеству методов во всех классах системы.

NOM = 6

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

Метрика равна общему количеству классов в системе.

NC = 1

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

Метрика равна общему числу строк кода системы.

LOCsum =40

**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 |
| interface ILogExport | string FilterName | void ExportToFile(string path, IEnumerable<Message> messages); |

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

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

|  |  |  |
| --- | --- | --- |
| 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 |
|  | | |

Рисунок иерархия классов client

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 interface ILogExport=1

WNC class LogExportToTXT=1

WNC class LogExportToXML=1

WNC class LogExporter =3

общий WNC=2.8

**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=0.4

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=4

СВО interface ILogExport=0

СВО class LogExportToTXT=1

СВО class LogExportToXML=1

СВО class LogExporter =1

СВО=1,4

**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+6=14

RFC interface ILogExport=2+0=2

RFC class LogExportToTXT=3+4=7

RFC class LogExportToXML=3+1=4

RFC class LogExporter =3+3=6

RFC =6.6

**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.

LCOM class Form1=38-6=32

LCOM interface ILogExport=2-0=2

LCOM class LogExportToTXT=3-0=3

LCOM class LogExportToXML=3-0=3

LCOM class LogExporter =6-2=4

LCOM = 8.8

**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.

CS class Form1=8+3=11

CS interface ILogExport=1+1=2

CS class LogExportToTXT=1+1=2

CS class LogExportToXML=1+1=2

CS class LogExporter =2+1=3

CS =4

**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.

NOO class LogExportToTXT=1

NOO class LogExportToXML=1

NOO=0.4

**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.

SI class LogExportToTXT=0.5

SI class LogExportToXML=0.5

SI = (NOO \* уровень) /Mобщ=0.2

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

В качестве индикатора размера может использоваться количество строк программы.

OSAVG class Form1=48/8=6

OSAVG interface ILogExport=2/2=1

OSAVG class LogExportToTXT=10/2=5

OSAVG class LogExportToXML=11/2=5.5

OSAVG class LogExporter =15/3=5

OSAVG=4.5

**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.

NPAVG class Form1=4/8=0.5

NPAVG interface ILogExport=1/2=0.5

NPAVG class LogExportToTXT=0/2=0

NPAVG class LogExportToXML=0/2=0

NPAVG class LogExporter =1/3=0.5

NPAVG =0.2666

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

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

DITАИС = 1

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

Метрика равна количеству методов во всех классах системы.

NOMАИС = 16

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

Метрика равна общему количеству классов в системе.

NCАИС = 5

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

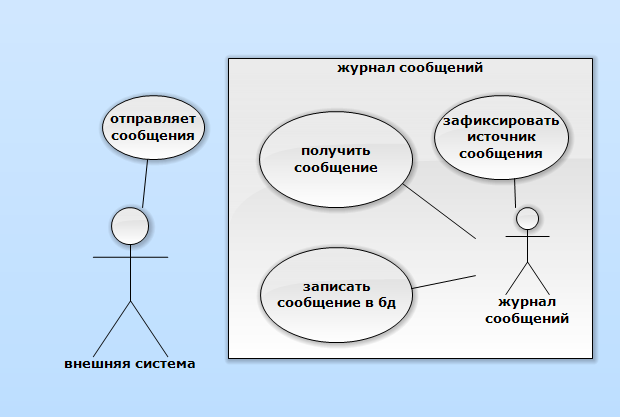
Метрика равна общему числу строк кода системы.

LOCsumАИС =76

## 2 Студент

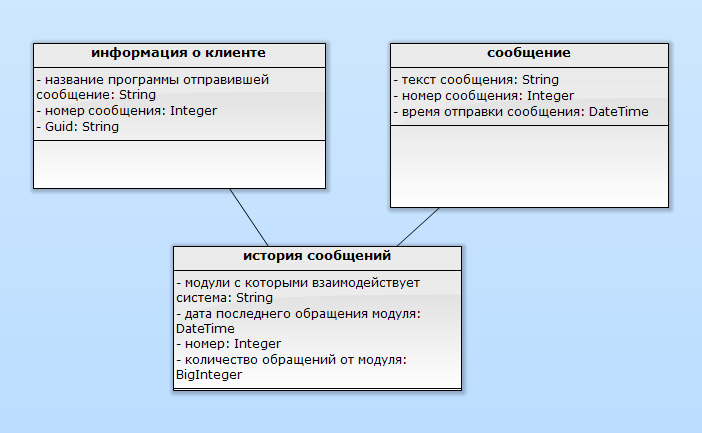
### Полностью определить и реализовать прецеденты (по варианту).

Диаграмма прецедентов для 2 студента.

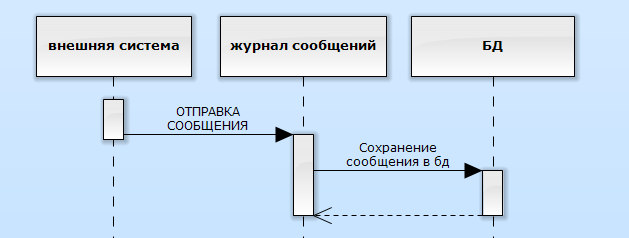


1. Диаграмма(ы) классов подсистемы,

Описание БД системы журнала сообщений



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

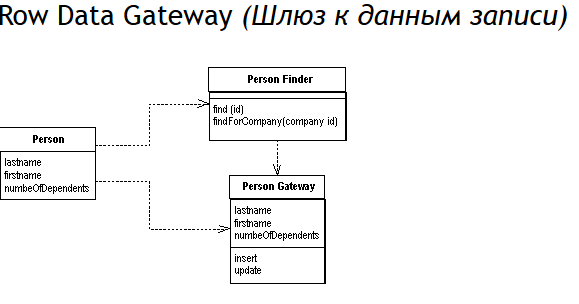


1. Диаграммы классов и последовательностей для иллюстрации примененных паттернов.

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.

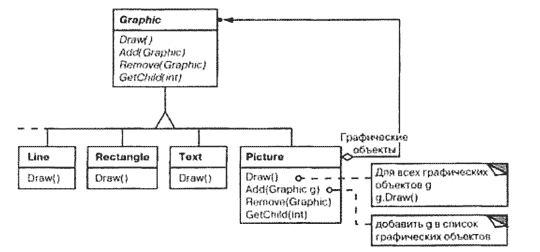
Встраивание кода доступа к БД в объекты, хранящиеся в памяти, может привести в некоторым неудобствам. Во-первых, если ваши объекты содержат бизнес-логику, добавление взаимодействия с БД увеличит сложность. Вскоре тестирование станет неудобным, если объекты, хранящиеся в памяти, завязаны на БД. Тесты 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 | WinHistoryDbContext db  ClientInfo Client | 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 |

**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 WinHistory =11

WNC class ClientInfo =2

WNC class Message =2

WNC class WinHistoryDbContext =3

общий WNC=4.5

**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=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 WinHistory =4

СВО class ClientInfo =0

СВО class Message =1

СВО class WinHistoryDbContext =2

общий СВО =1.75

**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 WinHistory =11+1=12

RFC class ClientInfo =2+0=2

RFC class Message =2+0=2

RFC class WinHistoryDbContext =3+1=4

RFC =5

**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.

LCOM class WinHistory =11-4=7

LCOM class ClientInfo =2-0=2

LCOM class Message =2-0=2

LCOM class WinHistoryDbContext =3-0=3

LCOM = 3.5

**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.

CS class WinHistory =8+2=10

CS class ClientInfo =2+0=2

CS class Message =2+0=2

CS class WinHistoryDbContext =3+1=4

CS =4.5

**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.

NOO=0

**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.

SI = 0

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

В качестве индикатора размера может использоваться количество строк программы.

OSAVG CS class WinHistory =57/11=5.1818

OSAVG ClientInfo =15/2=7.5

OSAVG Message =13/2=6.5

OSAVG class WinHistoryDbContext =17/3=5.666

OSAVG=6.2

**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.

NPAVG class WinHistory =2/8=0,25

NPAVG class ClientInfo =3/2=1.5

NPAVG class Message =5/2=2.5

NPAVG class WinHistoryDbContext =2/3=0,6

NPAVG =1.2666

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

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

DIT = 0

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

Метрика равна количеству методов во всех классах системы.

NOM = 18

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

Метрика равна общему количеству классов в системе.

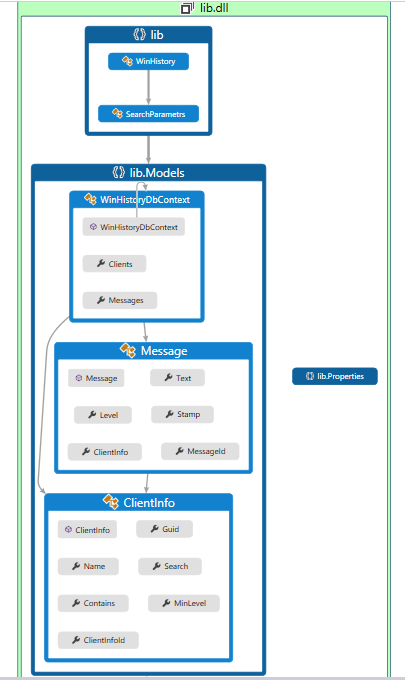
NC = 4

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

Метрика равна общему числу строк кода системы.

LOCsumАИС =102

**Итерация2**



рисунок

|  |  |  |
| --- | --- | --- |
| Класс | поля | методы |
| WinHistory | private static WinHistoryDbContext db | private WinHistory() |
| WinHistory(Guid guid, string name) |
| public static WinHistory Login(Guid guid, string name) |
| public void Send(string message, int level) |
| private readonly ClientInfo Client; | 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 |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| abstract class SearchParametrs | public List<SearchParametrs> Children | public SearchParametrs() |
| public void AddChild(SearchParametrs child) |
| public abstract IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs); |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| HasGuidParam | public Guid? HasGuid | public HasGuidParam(Guid guid) |
| public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs) |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| HasMinimumLevelParam | public Guid? HasGuid | public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs) |
| public HasMinimumLevelParam(int level) |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| NoParam |  | public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs) |

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| ContainsParam | string Contains | public ContainsParam(string str) |
| public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs) |

**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 WinHistory =11

WNC class ClientInfo =2

WNC class Message =2

WNC class WinHistoryDbContext =3

WNC SearchParametrs=3

WNC HasGuidParam=1

WNC HasMinimumLevelParam=1

WNC NoParam=0

WNC ContainsParam=1

общий WNC=2.6

**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 SearchParametrs=4

NOC=0.444

**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 WinHistory =4

СВО class ClientInfo =0

СВО class Message =1

СВО class WinHistoryDbContext =2

СВО SearchParametrs=1

СВО HasGuidParam=1

СВО HasMinimumLevelParam=1

СВО NoParam=1

СВО ContainsParam=1

общий СВО =1.3333

**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 WinHistory =11+2=13

RFC class ClientInfo =2+0=2

RFC class Message =2+0=2

RFC class WinHistoryDbContext =3+1=4

RFC SearchParametrs=3+0=3

RFC HasGuidParam=1+1=2

RFC HasMinimumLevelParam=1+1=2

RFC NoParam=1+0=1

RFC ContainsParam=1+1=2

RFC =3.444

**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.

LCOM class WinHistory =11-4=7

LCOM class ClientInfo =2-0=2

LCOM class Message =2-0=2

LCOM class WinHistoryDbContext =3-0=3

LCOM SearchParametrs=3-0=3

LCOM HasGuidParam=1-0=1

LCOM HasMinimumLevelParam=1-0=1

LCOM NoParam=1-0=1

LCOM ContainsParam=1-0=1

LCOM = 2.333

**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.

CS class WinHistory =11+2=13

CS class ClientInfo =2+0=2

CS class Message =2+0=2

CS class WinHistoryDbContext =3+1=4

CS SearchParametrs=3+1=4

CS HasGuidParam=2+1=3

CS HasMinimumLevelParam=2+1=3

CS NoParam=1+0=1

CS ContainsParam=2+1=3

CS =3.8

**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.

NOO HasGuidParam=1

NOO HasMinimumLevelParam=1

NOO NoParam=1

NOO ContainsParam=1

NOO=0.4444

**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 HasGuidParam=1

NOA HasMinimumLevelParam=1

NOA NoParam=1

NOA ContainsParam=1

NOA=0.444

**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.

SI HasGuidParam=0.5

SI HasMinimumLevelParam=0.5

SI NoParam=0.5

SI ContainsParam=0.5

SI =0.2222

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

В качестве индикатора размера может использоваться количество строк программы.

OSAVG CS class WinHistory =57/11=5.1818

OSAVG ClientInfo =15/2=7.5

OSAVG Message =13/2=6.5

OSAVG class WinHistoryDbContext =17/3=5.666

OSAVG SearchParametrs=9/3=3

OSAVG HasGuidParam=8/2=4

OSAVG HasMinimumLevelParam=8/2=4

OSAVG NoParam=5/1=5

OSAVG ContainsParam=8/2=4

OSAVG=5

**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.

NPAVG class WinHistory =2/8=0,25

NPAVG class ClientInfo =3/2=1.5

NPAVG class Message =5/2=2.5

NPAVG class WinHistoryDbContext =2/3=0,6

NPAVG SearchParametrs=1/3

NPAVG HasGuidParam=1/2=0.5

NPAVG HasMinimumLevelParam=1/2=0.5

NPAVG NoParam=0/1=0

NPAVG ContainsParam=1/2=0.5

NPAVG =0.742

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

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

DIT = 1

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

Метрика равна количеству методов во всех классах системы.

NOM = 24

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

Метрика равна общему количеству классов в системе.

NC = 9

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

Метрика равна общему числу строк кода системы.

LOCsumАИС =140

**Тестирование и отладка**

структурное

Основные понятия и принципы тестирования ПО Тестирование — процесс выполнения программы с целью обнаружения ошибок. Шаги процесса задаются тестами. Каждый тест определяет:

• свой набор исходных данных и условий для запуска программы;

• набор ожидаемых результатов работы программы.

### Функциональное

Функциональное тестирование — это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Тестирование «черного ящика» обеспечивает поиск следующих категорий ошибок:

1) некорректных или отсутствующих функций;

2) ошибок интерфейса;

3) ошибок во внешних структурах данных или в доступе к внешней базе данных;

4) ошибок характеристик (необходимая емкость памяти и т. д.);

5) ошибок инициализации и завершения.

Функциональное тестирование осуществлялось в процессе разработки, основную логику утилиты реализует библиотека lib.dll, а вспомогательную client.exe, который в свою очередь использует все функции предоставляемые lib.dll. Таким образом функциональное тестирование производилось проверкой корректных данных или реакций на действия, во вспомогательной утилите client.exe.

### Нагрузочное

Тестирование проводилось с помощью экземпляров нескольких программ text.exe(3),которые были модифицированны таким образом, что в бесконечном цикле без пауз посылали сообщения. Библиотека оставалась работоспособной, при незначительном снижении скорости. Тестирование осуществлялось на компьютере с процессором Intel Core i7 2650, 8Gb оперативной памяти, SSD диск хранения.

# Этап внедрения

## Системные требования

Требования для использования утилиты: ОС Windows7 и новее, .Net Fremework 4.5, Microsoft Sql Server Compact, наличие среды разработки .NET приложений.

## Установка

Необходимо скопировать папку с приложением целиком, после чего подключить lib.dll к проекту ,в котором предполагается использовать логировщик. После чего с помощью функции lib.WinHistory.Login установить соединение с библиотекой. Для просмотра накопленной истории следует использовать утилиту client.exe.

## Документация

В комплекте поставки включён chm файл, содержаний в себе полное описание методов, которые может использовать прикладной программист при работе с данной утилитой, основным документов описывающим работу утилиты является данный chm файл.

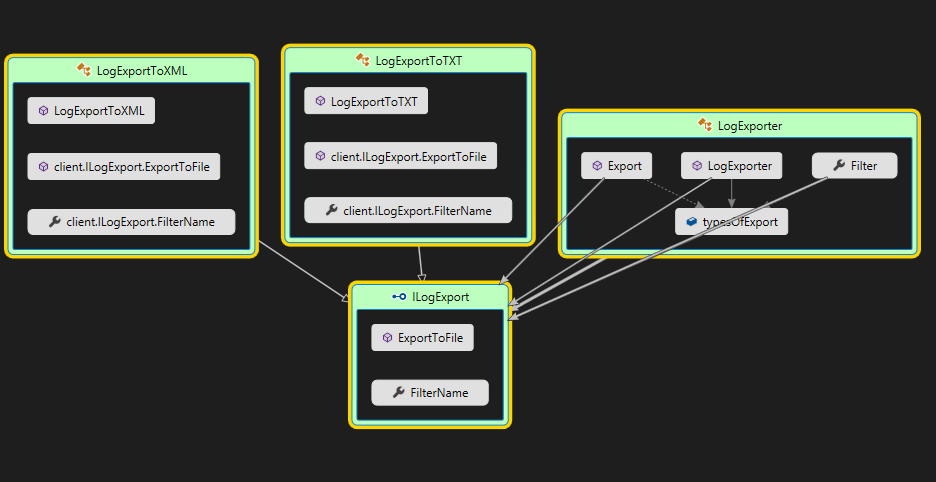
## Требования по внедрению

При внедрении следует учитывать факт,что целевой адиторией данного продукта являются программисты, поэтому предъявляется список требований к программисту: знание WinApi на базовом уровне, умение программировать на языка .NET платформы, знание технологии LINQ.

# Приложение А

Паттерны

1. Паттерн мост



* 1. Основной код реализации

Файл Exports.cs

public interface ILogExport

{

/// <summary>

/// Строка имени фильтра в диалоге сохранения окна.

/// </summary>

string FilterName { get; }

/// <summary>

/// Прозводит экспорт в необходимый формат

/// </summary>

/// <param name="path">Путь куда будет сохраняться файл.</param>

/// <param name="messages">Сообщения,которые были экспортированны.</param>

void ExportToFile(string path, IEnumerable<Message> messages);

}

/// <summary>

/// Реализация экспорта в текстовый формат

/// </summary>

class LogExportToTXT : ILogExport

{

string ILogExport.FilterName

{

get { return "Текстовый файл |\*.txt"; }

}

void ILogExport.ExportToFile(string path, IEnumerable<Message> messages)

{

try

{

var file = File.CreateText(path);

foreach (var msg in messages)

{

file.WriteLine(msg.Stamp + " " + msg.ClientInfo.Name + " " + msg.Level + " " + msg.Text);

}

file.Close();

}

finally

{

}

}

}

/// <summary>

/// Реализация экспорта в XML формат

/// </summary>

class LogExportToXML : ILogExport

{

string ILogExport.FilterName

{

get { return "XML Файл |\*.xml"; }

}

void ILogExport.ExportToFile(string path, IEnumerable<Message> messages)

{

try

{

var file = File.CreateText(path);

var ser = new System.Xml.Serialization.XmlSerializer(typeof(Message));

foreach (var msg in messages)

{

ser.Serialize(file, msg);

}

file.Close();

}

finally

{

}

}

} }

/// <summary>

/// Мост организующий экспорт в любой из доступных форматов.

/// </summary>

public class LogExporter

{

private List<ILogExport> typesOfExport;

/// <summary>

///

/// </summary>

/// <param name="types"> Список ссылок на объекты, реализовывающие экспорт</param>

public LogExporter(List<ILogExport> types)

{

typesOfExport = types;

}

/// <summary>

/// Список фильтров,которые появятся в диалоге сохранения файла.

/// </summary>

public string Filter

{

get

{

string s = "";

for (var i = 0; i < typesOfExport.Count; i++)

{

if (i + 1 == typesOfExport.Count)

{

s += typesOfExport[i].FilterName;

}

else

{

s += typesOfExport[i].FilterName + "|";

}

}

s = s.PadLeft(s.Length - 1);

return s;

}

}

/// <summary>

/// Производит экспорт сообщений в указанный формат

/// </summary>

/// <param name="path">Имя файла(полный путь) куда будет произведен экспорт</param>

/// <param name="index">Индекс фильтраэ, который был применен при сохранении</param>

/// <param name="messages"> Сообщения,которые будут экспортированны</param>

public void Export(string path, int index, IEnumerable<Message> messages)

{

typesOfExport[index].ExportToFile(path, messages);

}

}

* 1. Код использования

Файл Form1.cs

private void exportButton\_Click(object sender, EventArgs e)

{

var saveFile = new SaveFileDialog();

var exportTypes = new List<ILogExport> { new LogExportToTXT(), new LogExportToXML() };

var exporter = new LogExporter(exportTypes);

saveFile.Filter = exporter.Filter;

var dialogResult = saveFile.ShowDialog();

if (dialogResult == DialogResult.OK)

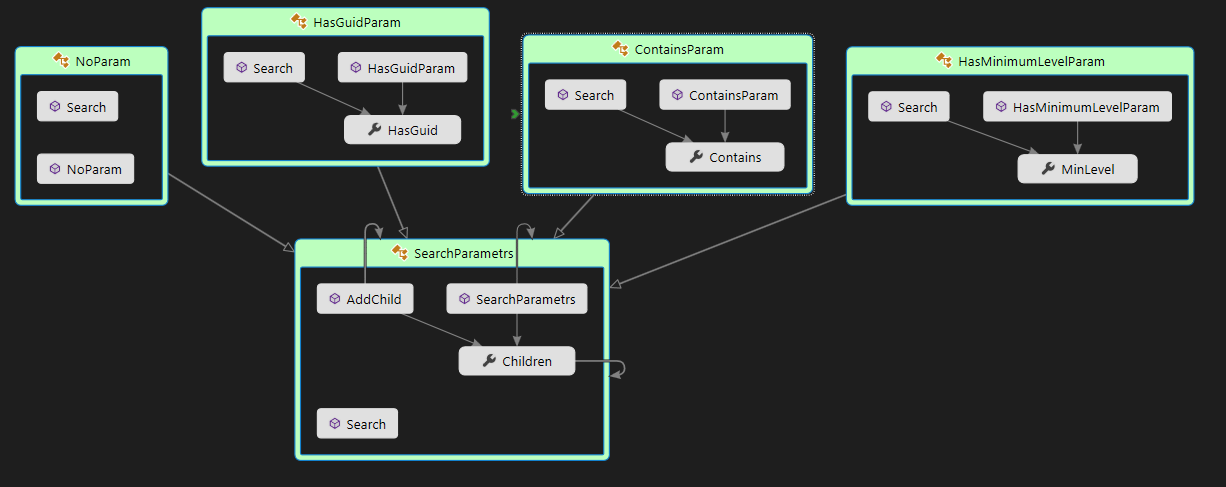
{

exporter.Export(saveFile.FileName, saveFile.FilterIndex - 1, (List<lib.Models.Message>)MessagesView.DataSource);

}

}

1. Паттерн «Компоновщик»



* 1. Основной код реализации

Файл SearchParametrsTree.cs

/// <summary>

/// Описывает параметры,которые могут быть применены для фильтрации IEnumerable Model.Message

/// </summary>

public abstract class SearchParametrs

{

/// <summary>

/// Список детей параметра, каждый каждый уровень вниз ,добавляет логическое и, вширь - логическое или

/// </summary>

public List<SearchParametrs> Children { get; private set; }

public SearchParametrs()

{

Children = new List<SearchParametrs>();

}

//public SearchParametrs Parent;

/// <summary>

/// Добавляет дочеренее древо параметров

/// </summary>

/// <param name="child">Новое дочернее древо параметров</param>

public void AddChild(SearchParametrs child)

{

//child.Parent = this;

Children.Add(child);

}

public abstract IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs);

}

public class HasGuidParam:SearchParametrs

{

/// <summary>

/// Имеет указанный Guid

/// </summary>

public Guid? HasGuid { get; set; }

public HasGuidParam(Guid guid)

{

HasGuid = guid;

}

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return from m in raw\_msgs where m.ClientInfo.Guid == HasGuid.ToString() select m;

}

}

public class HasMinimumLevelParam:SearchParametrs

{

/// <summary>

/// Минимальный уровень

/// </summary>

public int MinLevel { get; set; }

public HasMinimumLevelParam(int level)

{

MinLevel= level;

}

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return from m in raw\_msgs where m.Level >= MinLevel select m;

}

}

public class NoParam:SearchParametrs

{

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return raw\_msgs;

}

}

public class ContainsParam:SearchParametrs

{

/// <summary>

/// Содержит подстроку

/// </summary>

public string Contains { get; set; }

public ContainsParam(string str)

{

Contains = str;

}

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return from m in raw\_msgs where m.Text.Contains(Contains) select m;

}

}

1. Паттерн Модуль таблицы

В данной реализации модулем таблицы является атрибут объект db класса WinHistory.

Файл WinHistory.cs

public class WinHistory

{

private static WinHistoryDbContext db = new WinHistoryDbContext();

...

}

Через данную сущность мы получаем доступ ко всем таблицам бд . Разделение области определения структуры таблицы происходит в классах неймспейса lib.Models

1. Паттерн Data Mapper

Как и в случае с паттерном Модуль таблицы , Entity Framework реализует данный паттерн, изолируя объекты от бд.

Пример использования:

Файл WinHistory.cs

/// <summary>

///

/// </summary>

/// <param name="guid">Уникальный guid программы</param>

/// <returns>Возвращает информацию о программе клиенте </returns>

public ClientInfo ReciveClientInfo(Guid guid)

{

return (from clients in db.Clients where clients.Guid == guid.ToString() select clients).Single();

}

Здесь объект db.Clients позволяет обращаться с таблицей данных Clients как с объектом.

1. Паттерн «Шлюз к записи данных»

Возвращаемый объект ClientInfo предыдущего примера является ярким примером шлоюза к записи данных. Мы можем обращаться к данным записи в базе, как к обычному объекту, используя данный паттерн.

# Литература

1. Орлов С.А. Технологии разработки программного обеспечения. - СПб: Питер, - 2002 г. (Материалы сервера [http://wmate.ru](http://wmate.ru/) )
2. Якобсон А, Дуч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. - Спб.: Питер. - 2002 г.
3. Виноградова М.В. Курс лекций по дисциплине «Технологии разработки ПО»