Министерство образования и науки республики Казахстан
Петропавловский колледж железнодорожного транспорта

РАЗРАБОТКА СИСТЕМЫ КАТОЛОГИЗАЦИИ АНИМЕ КОНТЕНТА

ПКЖТ.1304.002.ДП.ПЗ.008.15

Выполнил:
Снежко Д.В.
_____ *ПО-41*
“____” _____ _____

2015

TO DO LIST

а) Актуальность +

б) Цель дипломного проекта +/-

в) Задачи дипломного проекта

г) Практическая значимость

д) Объект исследования

е) Характеристика вычислительной техники

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задачи каталогизации различного контента возникают повсеместно, от школьных библиотек до государственных документов, однако в таких программах заинтересованы не только крупные учреждения, но и рядовые пользователи персональных компьютеров, одна из таких аудиторий - фанаты аниме - жанра японской мультиплекации, имеющей очень большую популярность не только в Японии, откуда они родом, но и во всём мире, причём возростной порог уадитории колеблется от дошкольного до, вполне, зрелого 35-40 лет.

Как правило на счету зрителей аниме, не один десяток просмотреных мультфильмов, в связи с чем - возникают определённые потребности, такие как: ведение списка, которым можно поделиться с друзьями, или записать в него те аниме которые хочешь посмотреть, отмечать на какой серии остановился, а так же некоторые другие и разумеется, когда речь идёт не об одном десятке таких записей, возникает потребность иметь навигацию по этому списку, такие базовые вещи, как, поиск, фильтрация, добавление описания и картинки для большего удобства поиска и освежения воспоминаний при поиске, т.к. человеческий мозг наиболее эффективно работает с графической информацией, одного взгляда на картинку может хватить чтобы вспомнить (сюжет) аниме нежели просто прочитав название или вчитываясь в длинное описание.

Целью дипломного проекта является разработка программного продукта позволяющего каталогизировать аниме-контент, а так же связанные с ним, близкие по смыслу: мангу, AMV и Дорамы.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ предметной области

1.1.1 Общая характеристика объекта исследования

Каталогизация — это совокупность методов и процессов направленных на хранение, упорядочевание и классификацию какой-либо информации. Термин «каталогизация» употреблялся уже XVI веке. В XIX веке каталогизация становится научной дисциплиной, имеющей несколько самостоятельных направлений.

Аниме — японская анимация. В отличие от мультфильмов других стран, предназначенных в основном для просмотра детьми, бóльшая часть выпускаемого аниме рассчитана на подростковую и взрослую аудитории, и во многом за счёт этого имеет высокую популярность в мире.

Аниме отличается характерной манерой отрисовки персонажей и фонов. Издаётся в форме телевизионных сериалов, а также фильмов, распространяемых на видеоносителях или предназначенных для кинопоказа. Сюжеты могут описывать множество персонажей, отличаться разнообразием мест и эпох, жанров и стилей. Источниками для сюжета аниме-сериалов чаще всего являются: манга (японские комиксы), ранобэ (лайт-новел), или компьютерные игры (как правило, в жанре «визуальный роман»).

При экранизации обычно сохраняется графический стиль и другие особенности оригинала. Реже используются другие источники, например, произведения классической литературы. Есть также аниме, имеющие полностью оригинальный сюжет (в этом случае уже само аниме может послужить источником для создания по нему книжных и манга-версий). Значение термина «аниме» может варьироваться в зависимости от контекста. В западных странах аниме является объектом исследования учёных-культурологов, социологов и антропологов — Эри Идзавы, Скотта Маклауда, Сьюзан Напьер, Шерон Кинселлы и других.

1.1.2 Организация предметной области

Предметной областью являются рядовые пользователи персональных компьютеров

Рассмотрим процесс составления списка просмотренных аниме на примере обычного, среднестатистического пользователя ПК с использованием текстового редактора.

Процесс добавления записей в текстовый редактор начинается с запуска самого редактора, последующего поиска необходимого файла в файловой системе и его открытия, после чего пользователю отображается список из названий аниме, составленный им, список не имеет структуры и никак не помогает пользователю искать в нём интересующее его аниме, отмечать количество просмотренных серий, читать описание или поделиться с другом, добавление новой записи является самым простым процессом.

Таким образом можно заключить что текстовый редактор является плохим вариантом для выполнения поставленной задачи, далее предлагается рассмотрение другой программы.

Рассмотрим тот же процесс на другом программном продукте, например - тебличный процессор. Примером табличных процессоров являются такие программы как: Microsoft Excel, LibreOffice Calc, OpenOffice Calc, Calligra Sheets и множество других менее популярных продуктов.

Процесс добавления записей в табличный процессор является всё тем же поиском файл в файловой системе и его открытия, но в подобных продуктах, уже предлагается операция открытия «недавних» файлов, однако так как эти продукты рассчитаны на широкий круг задач, список таких файлов может быть слишком большим и не сохраняться полностью. После открытия файла пользователю предлагается таблица, где строка будет являться одной записью, а столбцы могут иметь дополнительное необходимое пользователю предназначение, например: год выпуска, студия, оригинальное название(не перевод), и прочее. В табличных процессорах уже имеется возможность сортировать список по тем или иным столбцам, что облегчает задачу поиска. Добавление новых записей всё так же требует самостоятельного(ручного) ввода всех данных, что является утомительным процессом и не исключающим ошибки и опечатки.

Теперь рассмотрим процесс каталогизации аниме в программе Database Anime.

Процесс работы с программой начинается с её запуска, при запуске программы, база данных уже будет загружена и пользователю представляется возможность работы с ней.

Добавление новой записи начинается с выбора необходимого раздела, если он ещё не выбран, при повторном использовании программы, запоминается на каком разделе пользователь завершил работу, после выбора раздела необходимо нажать на кнопку добавления новой записи, в следствие чего открывается диалоговое окно с предлагаемыми к заполнению полями. Пользователь может заполнять поля вручную, но это не самый интересный процесс, в программе предусмотрено автоматическое заполение данных, при наличии доступа к интернет, для этого требуется начать вводить название, в процессе его ввода пользователю предоставится выпадающий список с найденными на сервере записями имеющими в названии последовательность введённых пользователем символов, в том числе на русском языке, для выбора достаточно стрелочками мыши выбрать нужный пункт и нажать клавишу ввода, либо выбрать пункт мышью и нажать на кнопку поиска, в результате проделанных дейцствий данные будут загружены с сервера и заполены в соответствующие поля.

Процесс навигации по каталогу является максимально простой и удобной операцией, список с названиями отображается в левой части главного окна. Сверху списка находится выпадающий список для выбора раздела, которых в программе несколько, а под ним список фильтров специфичных для каждого раздела. Пользователь может сортировать список по алфавиту, году выпуска либо по дате добавления в каталог; фильтровать его по различным предлагаемым фильтрам и производить поиск по названию. Для выбора пункта достаточно кликнуть по нему мышью, в результате чего в правой части окна отобразятся данные выбранной пользователем записи.

1.2 Анализ аналогов программного продукта

1.2.1 AnimeList

Аналогичным по назначению и функционалу продуктом язвляется программа AnimeList, первая версия которой была выпущена Владиславом Куртуков, 08.11.2011 которой была заброшена в 2014 году, при этом программный продукт сохранил множество недостатков, которые остались неисправленными, а так же спустя некоторое время, некоторый функционал программы, связанный с автоматическим заполнением контента стал неработоспособным, причиной тому послужил неправильный подход к данной задаче, что привело ко множеству ошибок в будущем.

1.2.2 Основные преимущества Database Anime

- Стабильность, программа тщательно тестируется перед выпуском версий.
- Open Source и свободная лицензия, пользователи могут свободно делиться программой с друзьями, а так же другие программисты могут принять участие в разработке программы и даже написать собственную программу основываясь на кодовой базе Database Anime, с одним лишь обязательством - она должна быть опубликована под той же лицензией.
- Кроссплатформенность, программа работает на 3х основных платформах - Windows, Linux, MacOS. {Хотя для последнего пока ещё нет официальной сборки.}
- Эргономичный дизайн и возможность его модификации благодаря настройкам программы, пользователи могут свободно менять и настраивать его.
- Не требует нарушения политики безопасности операционной системы. Примером такового является запуск от имени администратора в операционных системах семейства Microsoft Windows.

– Интернационализация. Официально предоставляется русская и английская локализации, однако ничто не мешает добавить новую.

– Техническая поддержка, предоставляется на официальной странице программы Вконтакте.

– Имеется конвертер из экспортного формата программы AnimeList

– Автономность. Для работы программы не требуется постоянное соединение с интернетом, все данные хранятся на локальном компьютере пользователя, что так же означает что пользователь может легко перенести свою БД на любой другой компьютер.

1.3 Структура и функционирование системы

Разрабатываемый программный продукт позволяет хранить и отображать список аниме, манги, AMV и дорамы в удобном виде, благодаря функциям поиска и фильтрации заметно упрощается навигация и поиск необходимых пользователю записей. Все данные храняться в базе данных что обеспечивает быстрый доступ даже при больших объемах информации. При работе пользователю не потребуется каждый раз заново открывать один и тот же файл, что упростит работу с программой. Каждый пользователь операционной системы имеет собственную базу данных и собственные настройки которые никак не конфликтуют друг-с-другом. В программе предусмотрено четыре раздела что осложняет работу, для решения этой проблемы раздел на котором пользователь завершил работу с программой будет автоматически открыт при её запуске что уменьшает излишние действия пользователя.

1.4 Функции разрабатываемой системы

Приложение реализует следующий функционал:

– Добавление записей - ползователи программы могут добавлять записи в один из четырёх разделов.

– Просмотр записей добавленных пользователем, с отображением всей информации об аниме.

– Редактирование записей - редактирование записей, необходимо для внесения пользователем изменений, таких как смена обложнки и других частей. Для быстрого изменения колличества просмотренных серий, во время просмотра конкретной записи доступно прямое изменение колличества просмотренных серий используя элемент отображения прогресса просмотра.

– Удаление записей - позволяет удалить запись в том случае если она по каким-либо причинам больше не нужна пользователю.

– Сортировка и фильтрация записей по различным критериям.

– Автоматическое заполнение записей. Программа предлагает пользователю функцию автоматического заполнения записей, подсказывая пользователю возможные варианты во время ввода названия, выбрав из списка необходимое пользователю название, вся информация автоматически будет получена с сайта и соответствующие поля будут заполнены, после чего пользователь может отредактировать эту информацию, например указать сколько серий он просмотрел и сохранить запись.

– Проверка и уведомление о выходе новой версии. При запуске программа проверяет наличие новых версий в официальном репозитории проекта и выделяет версию программы красным цветом. Такой подход обусловлен своей ненавязчивостью и информативностью, так как в разных операционных системах имеются различные подходы к обновлению программного обеспечения.

– Импорт и экспорт - в программном продукте пользователю предлагаются возможности для экспортирования и импортирования всех своих записей в формат пригодный для переноса данных. Это может использоваться для переноса данных в другой программный продукт, или для того чтобы поделиться своими записями с другими пользователями, помимо этого это могут быть такие операции как: резервное копирование,

распространение информации непосредственно с контентом и прочих.

- Смена дизайна приложения - благодаря возможности смены дизайна пользователи имеют возможность настраивать внешний вид приложения в соответствии с их предпочтениями, а так же делиться с друзьями своими настройками дизайна.
- Локализация - в приложении предусмотрена возможность локализации интерфейса программы.
- Просмотр файловой системы - указывая определённую директорию во время добавления записи, пользователь активирует возможность просмотра этой директории во время просмотра соответствующей записи, что упрощает поиск и запуск видеоматериалов, позволяя делать это из приложения.
- Широкий спектр настроек - очень многие элементы программы пользователь имеет возможность настроить в соответствии с его предпочтениями.

1.5 Информационное и технологическое обоснование проекта

1.6 Инструментальное обоснование проекта

1.6.1 Эргономика программного обеспечения

Эргономика - Научная дисциплина, изучающая взаимодействие человека и других элементов системы, а также сфера деятельности по применению теории, принципов, данных и методов этой науки для обеспечения благополучия человека и оптимизации общей производительности системы.

Программный продукт должен быть разработан с учётом привычек пользователей и накопленного ими опыта, интерфейс программы не должен выбиваться из общего вида системных приложений но в то же время должен быть инновационным и лаконичным.

Пользователи должны иметь возможность настраивать отображение элементов управления программы и их размеры в соответствии с их нуждами предпочтениями, помимо этого пользователи должны иметь возможность смены стиля интерфейса.

В связи с тем что аниме-культура имеет широкое распространение по всему миру - интерфейс программы должен быть локализуемым на другие языки мира для охвата большей аудитории.

Помимо прочего в различных операционных системах, приссутствуют свои устоявшиеся нормы, такие как место на файловой сестеме, где программы хранят свои данные, например, для операционной системы семейства Microsoft Windows таким местом является «C:\Users\<Имя_Пользователя>\AppData\», в то время как для дистрибутивов UNIX-подобных систем, таким местом является «/home/<Имя_Пользователя>/.local/share/». Данное обстоятельство, несомненно, так же относится к эргономичности развёртывания программ под различными операционными системами.

1.6.2 Язык программирования C++

Проект написан на языке C++, получившем широкое распространение.

C++ — компилируемый статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных

программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

1.7 Фреймворк Qt

1.7.1 Кросплатформенность

Фреймворк Qt является кроссплатформенным, что позволяет разрабатывать один программный продукт и распространять его для всех операционных систем не переписывая исходный код продукта. Для этого достаточно скомпилировать продукт под целевой платформой и собрать дистрибутив под эту платформу, например для операционной системы семейства Microsoft Windows это может быть установочный пакет .exe или .msi; для MacOS - .dmg; для дистрибутивов на базе ядра Linux - .deb, .rpm, .pkg и т.д.

Кроссплатформенное (межплатформенное) программное обеспечение — программное обеспечение, работающее более чем на одной аппаратной платформе и/или операционной системе. Типичным примером является программное обеспечение, предназначенное для работы в операционных системах Linux и Windows одновременно.

1.7.2 Интернационализация и локализация

Фреймворк Qt предоставляет отличные средства для интернационализации продуктов, упрощая процесс локализации продукта ко всем языкам до простой операции подготовки перевода всех текстов приложения на определённый язык. Таким образом разрабатываемый продукт может быть переведён на всеразличные языки мира. На данный момент в наличие имеется 2 локализации: русская и английская, но благодаря программным решениям добавление новой локализации в продукт, не требует полной перекомпиляции всей программы, а сводится к добавлению файла с соответствующей локализацией в одно из стандартных мест на файловой системе, после чего соответствующая локализация может быть выбрана и установлена в настройках приложения. Благодаря такой реализации, каждый пользователь операционной системы может добавить и установить для себя собственную локализацию не пребегая к помощи системного администратора.

Интернационализа́ция — технологические приёмы разработки, упрощающие адаптацию продукта (такого как программное или аппаратное обеспечение) к языковым и культурным особенностям региона (регионов), отличного от того, в котором разрабатывался продукт. Есть важное различие между интернационализацией и локализацией. Интернационализация — это адаптация продукта для потенциального использования практически в любом месте, в то время как локализация — это добавление специальных функций для использования в некотором определённом регионе. Интернационализация производится на начальных этапах разработки, в то время как локализация — для каждого целевого языка.

1.7.3 База дынных SQLite

Для хранения данных в приложении используется популярная база дынных SQLite, являющаяся простым но в тоже время достаточно мощьным средством, получившим широкое распространение, в качестве локальных баз данных приложения, не требующих многопользовательских подключений.

SQLite — компактная встраиваемая реляционная база данных. Исходный код библиотеки передан в общественное достояние. В 2005 году проект получил награду Google-O'Reilly Open Source Awards.

Слово «встраиваемый» означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

В частности SQLite используют такие проекты как:

- Skype;
- Mozilla Firefox;
- Android API;
- Фреймворк Qt.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Система контроля версий

Любая информациооная система процесс разработки которой не заканчивается на достижении кокого-либо конкретного результата, предполагает постоянную доработку и внесение изменений, иногда такие изменения могут повлечь за собой серьёзные изменения в кодовой базе проекта, выводящие его из функционального состояная на время разработки новых функциональных возможностей.

В связи с тем что, разработка новых функциональных возможностей программного продукта может продолжаться весьма продолжительное время, теряется возможность внесения критически важных обновлений в текущую версию продукта, исходя из этого становится очевидной необходимость каким-либо образом контролировать процесс разработки таким образом чтобы разработка новой версии продукта не вносила осложнений в процесс доработки текущей его версии.

Системы контроля версий предназначены, помимо всего прочего и для решения таких задач, они позволяют контроллировать процесс разработки программного продукта, имея возможность создавать контрольные точки, такие как выход новой версии и в последствии возможность возврата к любой из контрольных точек, помимо этого системы контроля версий позволяют созавать различные ветви разработки программы, например, основная ветвь может постоянно отражать рабочее состояние продукта, а отдельная ветвь для критически важных обновлений позволит разрабатывать их не выводя из строя основную версию, причём в последствии эти ветви могут синхронизироваться друг с другом таким образом объеденяя изменения с различных ветвей.

Так же, системы контроля версий так же позволяют вести разработку программного продукта нескольким разработчикам, значительно упрощая процесс разработки благодаря возможности синхронизации(слияния) ветвей. Таким образом 2 разработчика могут работать над различными частями программного продукта, а по завершении (объеденить/синхронизировать/слить) изменения из своих ветвей в главную ветвь получая новую работоспособную версию продукта.

Существуют централизованные и децентрализованные системы контроля версий, основополагающим различием которых является расположение основно-

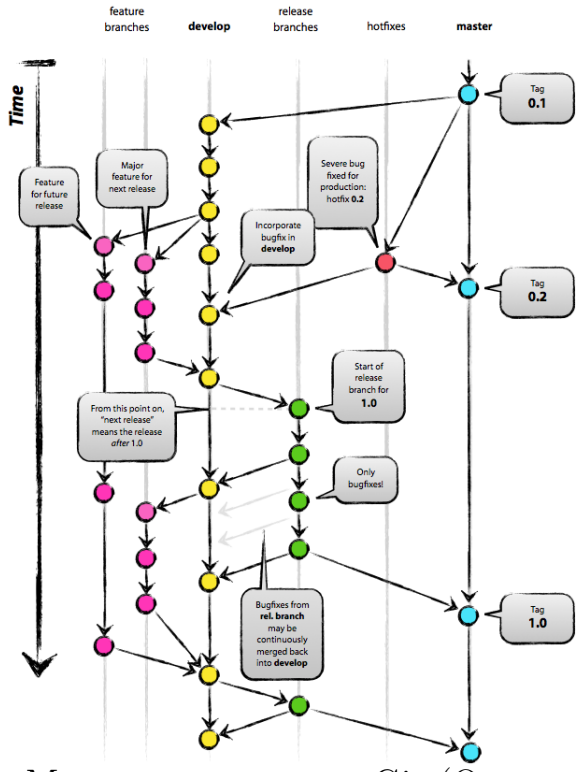

Рисунок 2.1 – Модель ветвления в Git (Оригинальная идея)

го репозитория. В централизованных системах контроля версий существует один единственный репозиторий с которым работают все разработчики и все изменения отправляются туда, минусом такого подходя является зависимость разработчика от доступа к основному репозиторию, а так же проблемы при слиянии(синхронизации) ветвей, в централизованных системах контроля версий этот процесс абсолютно всегда является болезненным, напротив же в децентрализованных системах контроля версий, ярким представителем которых является Git, процесс слияния(синхронизации) различных ветвей является обычным делом, в таких системах каждый разработчик имеет свою копию репозитория, с которой может распоряжаться как угодно и даже если он внесёт какие-либо изменения приносящие вред всему проекту или выходу из строя самой системы контроля версий - эти изменения окажут влияние только на его копию проекта, таким образом разработчик может протестировать внесённые изменения на своём компьютере и убедится в их работоспособности, помимо того имея на руках копию репозитория он может работать из любого места, будь то самолёт, дача или метро не завися от наличия доступа к интернет.

Таким образом используя в своей работе систему контроля версий я имею возможность, гибко работать с проектом, не боясь потерять какие-либо измене-

ния или вывести из строя проект целиком, в любой момент я имею возможность вернуться к лююой из контрольных точек и внести изменения, которые затем с лёгкостью могу выполнить слияние с любой другой ветвью разработки. На рисунке 2.1 изображена модель ветвления которую я использую в процессе разработки.

2.2 Документирование

Важной частью любого серьёзного проекта является документация, в документации разработчик может и обязан зафиксировать все аспекты работы системы и отдельных её частей. Документация позволяет другим разработчикам узнать необходимую им информацию по интересующему их модулю системы и работе системы в целом, что облегчает другим разработчикам процесс вхождения в разработку проекта.

Многие хорошие проекты разработчики которых забросили их разработку и поддержку, из за отсутствия документации приходится переписывать с нуля, потому как никто не знает как устроена система и о том как взоимодействуют между собой её отдельные модули, благо что в последнее время о важности документации знают все и новые проекты разрабатываются уже с документацией, но некоторые даже крупные проекты не пишут толковую документацию, а делают это только для вида, например фремеворк FireMonkey, по которому очень трудно найти внятную документацию, но в некоторых проектах, например фремеворк Qt - документация настолько хороша что в ней можно найти ответ на любой возникающий вопрос, на любую мелочь, что является очень весомым плюсом данного фремеворка и сыграло немаловажную роль в моём престрастии к нему.

Потому как документирование является такой важной частью, в проекте задокументировано более половины всех классов и их методов используемых в программном продукте, на двух языках русском и английском языках.

```
/*! \~russian
 * \brief Метод для поиска аниме по названию
 * \param title - искомое название
 * \param limit - ограничение на количество получаемых данных
 */
/*! \~english
 * \brief Search of anime on the website by title
 * \param title - searching title
 * \param limit - is a limit of the response
 */
void ShikimoriApi::searchAnime(QString title, short limit)
{
    QNetworkAccessManager* manager = new QNetworkAccessManager(this);
    connect(manager, &QNetworkAccessManager::finished,
            this,    &ShikimoriApi::replyAnimeSearch);

    QUrl url = shikimoriUrl + "/api/animes?search=" + title + "&limit=" + QString::number(limit);
    manager->get( QNetworkRequest( url ) );
}
```

Рисунок 2.2 – Пример использования Doxygen

2.2.1 Doxygen

Для документирования в проекте применяется Doxygen, так как он является наиболее популярным программным продуктом предназначеным для этих задач. На рисунке 2.2 представлен пример того как выглядит документация к одному из методов в проекте.

Doxygen - это кроссплатформенная система документирования исходных текстов, которая на сегодняшний день, по имеющему основания заявлению разработчиков, стала фактически стандартом для документирования программного обеспечения, написанного на языке C++, а также получила пусть и менее широкое распространение и среди ряда других языков.

Doxygen генерирует документацию на основе набора специальным образом комментированных исходных текстов и также может быть настроен для извлечения структуры программы из недокументированных исходных кодов. Возможно составление графов зависимостей программных объектов, диаграмм классов и исходных кодов с гиперссылками. Результатом работы программы Doxygen является готовая документация для распространения и использования. Doxygen позволяет генерировать на основе исходного кода, содержащего комментарии специального вида, красивую и удобную документацию, содержащую в себе ссылки, диаграммы классов, вызовов и т.п. в различных форматах: HTML, LaTeX, CHM, RTF, PostScript, PDF, man-страницы.

Doxygen используется многими проектами, в том числе KDE, Pidgin,

Torque Game Engine, AbiWord, Mozilla, FOX toolkit, Crystal Space, Drupal. Есть встроенная поддержка в KDevelop.

2.3 Модульное тестирование

Любой современный программный продукт представляет собой сложную систему разбитую на некоторые отдельные модули, такое разложение системы на модули заложена в саму суть Объектно Ориентированного Программирования, где каждый класс является отдельным модулем, который так же может зависить от других модулей которые выполняют более мелкие его функции.

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Основаня идея модульного тестирования состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом.

Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных; в ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с базой данных и реализует этот интерфейс, используя свой собственный mock-объект. Это приводит к менее связанному коду, минимизируя зависимости в системе.

В проекте применяется технология модульного тестирования, что упрощает процесс тестирования, увеличивет надёжность и отказоустойчивость программы, благодаря возможности отслеживать некорректное поведение классов сразу на этапе разработки, когда их исправление является наиболее простым и эффективным.

2.4 Качество кода

2.4.1 Паттёрны проектирования

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться. «Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные. На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы.

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению свое имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования

В поекте воплощены такие шаблоны проектирования как:

- Information Expert - Обработкой информации должен заниматься тот объект, который владеет этой информацией;
- Creator - Создавать экземпляры объектов должен тот класс который использует эти объекты;
- Low Coupling - Распределить обязанности между объектами так, чтобы степень связанности оставалась низкой;
- Hight Cohesion - Каждый класс должен отвечать только за одну функцию;

Помимо перечисленных паттёрнов GRASP, используется множество паттёрнов GoF

2.4.2 UML

Для проектирования новых модулей, я применяю UML диаграммы, что позволяет мне спроектировать класс или систему классов, до того как я приступлю к кодированию. Такой подход позволяет спроектировать систему и понять что потребуется от этой системы, как будет работать и взаимодействовать с внешними системами. Такие дааграммы так же строятся программой Doxygen на основе

имеющихся классов и применяются в документации, как наглядная модель иллюстрирующая взаимосвязь компонентов.

UML (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования.

Преимущества UML

- UML объектно-ориентирован, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках;
- UML позволяет описать систему практически со всех возможных точек зрения и разные аспекты поведения системы;
- Диаграммы UML сравнительно просты для чтения после достаточно быстрого ознакомления с его синтаксисом;
- UML расширяет и позволяет вводить собственные текстовые и графические стереотипы, что способствует его применению не только в сфере программной инженерии;
- UML получил широкое распространение и динамично развивается.

2.4.3 Рефакторинг

В своём проекте я регулярно провожу рефакторинг кода, так как переодические исправления и добавление новго функционала, {загрязняют} код и делают его нечитабельным, а так же довольно часто приводят к последующим ошибкам. Переодически проводя рефакторинг кода, я улучшаю его отдельные части, а так же структуру проекта в целом, что позволяет легче добавлять нововведения и вносить изменения в уже существующий код.

В программировании термин рефакторинг означает изменение исходного кода программы без изменения его внешнего поведения. В экстремальном программировании и других гибких методологиях рефакторинг является неотъемлемой частью цикла разработки ПО: разработчики попеременно то создают новые тесты и функциональность, то выполняют рефакторинг кода для улучшения его логичности и прозрачности. Автоматическое юнит-тестирование позволяет убедиться, что рефакторинг не разрушил существующую функциональность.

Иногда под рефакторингом неправильно подразумевают коррекцию кода с заранее оговоренными правилами отступа, перевода строк, внесения комментариев и прочими визуально значимыми изменениями, которые никак не отражаются на процессе компиляции, с целью обеспечения лучшей читаемости кода.

Рефакторинг изначально не предназначен для исправления ошибок и добавления новой функциональности, он вообще не меняет поведение программного обеспечения и это помогает избежать ошибок и облегчить добавление функциональности. Он выполняется для улучшения понятности кода или изменения его структуры, для удаления «мёртвого кода» — всё это для того, чтобы в будущем код было легче поддерживать и развивать. В частности, добавление в программу нового поведения может оказаться сложным с существующей структурой — в этом случае разработчик может выполнить необходимый рефакторинг, а уже затем добавить новую функциональность.

Это может быть перемещение поля из одного класса в другой, вынесение фрагмента кода из метода и превращение его в самостоятельный метод или даже перемещение кода по иерархии классов. Каждый отдельный шаг может показаться элементарным, но совокупный эффект таких малых изменений в состоянии радикально улучшить проект или даже предотвратить распад плохо спроектированной программы.

2.4.4 Статический анализатор кода CppCheck

Для проведения статического анализа кода, в проекте применялась утилита CppCheck, это довольно молодой проект статического анализа с открытым исходным кодом, ориентированный в первую очерень на нахождение реальных

ошибок в коде с минимальным количеством ложнных срабатываний. Хотя это не самое мощьное средство, но при помощи него удалось найти уязвимость в проекте Xorg, которая существовала там почти 23 года, что является приемлемым результатом и вполне подходит для небольших проектов. Статический анализатор позволяет находить больше ошибок на этапе компиляции, что соответственно упрощает разработку и значительно повышает качество программного продукта.

Статический анализ кода — это технология поиска ошибок в программах путем разбора исходного кода и поиска в нем паттернов (шаблонов) известных ошибок. Эта технология реализуется специальными инструментами, называемыми статическими анализаторами кода.

Слово «статический» означает, что код разбирается без запуска программы на выполнение. Инструменты, которые анализируют программу во время ее работы, называются динамическими анализаторами кода.

Наиболее известные статические анализаторы выпускают компании Coverity, Klocwork, Gimpel Software. Популярные динамические анализаторы делают компании Intel (Intel Parallel Inspector) и Micro Focus (DevPartner Bounds Checker), а так же PVS-Studio.

Результат работы статического анализатора — это список обнаруженных в коде потенциальных проблем с указанием имени файла и конкретной строки. Другими словами, это список ошибок, очень похожий на тот, что выдает компилятор. Термин «потенциальные проблемы» используется здесь не случайно. К сожалению, статический анализатор не может абсолютно точно сказать, является ли эта потенциальная ошибка в коде реальной проблемой. Это может знать только программист.

Инструменты для статического анализа кода делятся по типу поддерживаемых языков программирования (Java, C#, C, C++), по диагностируемым проблемам (анализаторы общего назначения или специализированные, например, для разработки 64-битных или параллельных программ).

2.5 Описание программных компонентов системы

о всех виджетах (что использовались)

2.6 Документальное описание информационной системы

о назначении каждой кнопочки

СПИСОК ЛИТЕРАТУРЫ

1. Кнут Д.Э. Искусство программирования / перевод с англ. Козаченко Ю.В. в 3-х т. М.: Просвещение 1997.
2. Ульман Дж. Основы систем баз данных / Дж.Ульман, Д.Уидом. М.: Лори, 2000;
3. Шлее М. Qt 4.8 - Профессиональное программирование на C++, «БХВ-Петербург», 2012
4. Саммерфилд М. Qt Профессиональное программирование, разработка кросплатформенных приложений на C++, «Символ-Плюс», 2011
5. Фаулер М. Рефакторинг - Улучшение существующего кода, «Символ-Плюс», 2003
6. Банда Четырёх Приёмы объектно-ориентированного проектирования, «Питер», 2001
7. Скотт Чакон и Бен Страуб, Pro-Git, «Apress», 2009