1. Этапы построения проблемно-центрированного дизайна.

анализ задач и пользователей;

•

выбор репрезентативных задач;

•

заимствование;

•

черновое описание дизайна;

•

обдумывание дизайна;

•

прототипа;

создание макета или

•

пользователями;

тестирование дизайна с

•

итерирование;

•

реализация;

•

отслеживание эксплуатации;

•

изменение дизайна.

1. CWT анализ.

Пример – CD копировщик. Cognitive walkthrough. Формализованный способ представления мыслей и действий людей, когда они пользуются интерфейсом в первый раз. \*Все алгоритмы действий пользователя, как он их себе представляет должны давать результат\*. Идём по действиям алгоритма и помечаем как нужно улучшить интерфейс, чтобы \* этот пункт выполнялся. Дорлжна быть обратная связь от программы. Для CWT нужно сразу иметь список действий для решения задачи. Если пользователь не знает, что-то ещё д.б. сделано, то пусть программа это сделает сама.

1. GOMS анализ.

Goals Operations Methods Selections. Подсчёт времени выполнения опытным пользователем. Время затрачиваемое машиной на выполнение операций без участия человека не должно учитываться. Семантические единицы: К 0.2 с – нажатие клавиши, М 1.35 с – мыслительная деятельность, В 0.2 с – клик мышью, Р 1.1 с – point – указание мышью, R – ожидание ответной реакции компьютера, H 0.4 с – hover перенос руки с клавиатуры на мышь или наоборот, D - проведение с помощью мыши прямой линии (например, выделение или прокрутка текста);

Результат – оценка среднего времени , понимание того, какие улучшения можно провести.

1. Правила Нильсена-Молиха.
2. Простой и естественный диалог – информация только по теме
3. Говорите на языке пользователя
4. Минимизируйте загрузку памяти пользователя(введенная информация должна оставаться на экране как можно дольше.
5. Будьте последовательны(должна быть возможность изучить действия в одной части программы и применить их снова).
6. Обеспечьте обратную связь
7. Обеспечьте хорошо обозначенные выходы(из окон доп. Свойств например).
8. Обеспечьте быстрые клавиши и ярлыки
9. Хорошие(информативные) сообщения об ошибках
10. Предотвращайте ошибки(например при переключении калькулятора в двоичный режим можно отключить цифры 2-9).
11. Принципы организации графического интерфейса.

1)Принцип кластеризации(напр. кнопки для отдельных функций располагаются рядом в блоках с названиями)

2) «Видимость отражает полезность» - наиболее часто используемые элементы на самом видном месте

3)Принцип интеллнектуальной последовательности(экраны с похожими функциями д.б. похожи, напр. Кнопка «помощь» д.б. всегда в одном и том же месте

4) цвет в последнюю очередь (цвет не однозначен для разных культур

5) Принцип уменьшения беспорядка(одинаковые шрифты).

1. Общий алгоритм работы со списком QList.

Нежелательно использовать [ ], вместо этого лучше использовать at( )

QList<int> list;//объявили

list << 10 << 20 << 30;//внесли данные

Для перечисления элементов:

QValueList<int>::iterator it = list.begin();

while (it != list.end()) {

qDebug() << "Element:" << \*it;

++it;

}

move Перемещает элемент с одной позиции на другую

rernoveFirst Удаляет первый элемент списка

rernoveLast Удаляет последний элемент списка

-

swap Меняет местами два элемента на указанных позициях

---- --

takeAt Возвращает элемент на указанной позиции и удаляет его

~--

takeFirst Удаляет первый элемент и возвращает его

~----- -

takeLast Удаляет последний элемент и возвращает его

-- --------

toSet Возвращает контейнер QSet<T> с данными, содержащимися в объекте

QList<T>

~ --- - ----------

toStdList Возвращает стандартный список STL std:: list<T> с элементами,

содержащимися в объекте QList<T>

toVector Возвращает объект вектора QVector<T> с элементами, содержащимися в объекте QList<T>

1. Общий алгоритм работы с текстовыми файлами в Qt.

QFile file(“file.txt”);//Создали объект типа QFile.

if (file.open(QIODevice::WriteOnly))// Если записываемый файл не существует, он будет создан. В противном случае он будет перезаписан.

{

    file.write("Test string\n");

    file.write("Test string2");

    file.close();

}

if ((file.exists())&&(file.open(QIODevice::ReadOnly)))//построчное чтение

{

    QString str="";

    while(!file.atEnd())

    {

        str=str+file.readLine();

    }

    ui->textBrowser->setText(str);

    file.close();

}

if ((file.exists())&&(file.open(QIODevice::ReadOnly)))//считали сразу всё

{

    ui->textBrowser->setText(file.readAll());

    file.close();

}

*if*(file.open(QIODevice::Append)){ *//will* *write* *to* *the* *end*

QTextStream stream (&file);//создал поток, ввёл в него ссылку на QFile

QString str = "\na new line";//Создал строчку

stream<<str.toUpper();//ввёл в поток строчку, да ещё и прописными бёквами

*if*(stream.status()!=QTextStream::*Ok*){

qDebug()<<"Ошибка чтения файла";

}

file.close();

}

1. Механизм работы сигналов и слотов.

Создан для обеспечения ООП. СиС – средства, позволяющие эффективно производить обмен информацией о событиях, производимых объектами. Q\_OBJECT – необходимый макрос. Объект "телефон" выслал сигнал "звонок", на который объект "человек" отреагировал слотом "снятия трубки". Не явл. Частью С++. Сигнал объявлен в одном месте, функция им запускаемая в другом. Можно сделать pudlic void sendSignal(){emit doIt();} Т.о. сигнал doIt(); м.б. вызван другим классом. Слоты – методы, которые присоединяются к сигналам. Связь может производиться между разными потоками(объектами).

1. Менеджер компоновки: определение, виды, назначение.

Позволяет размещать виджеты/компоновки на поверхностях других виджетов или компоновок.

QHBoxLayout//это суётся в виджет, а может и в другой Layout быть засунуто

QVBoxLayout

QGridBoxLayout

QBoxLayout принимает параметры( leftToRight и т.п. для указания мест размещения)

# include <QtWidgets>

int main(int argc, c har\*\* argv )

{QApplication app(argc, argv);

QWidget wgt;

QPushButton\* pcmdA new QPushButton ("А");

QPushButton\* pcmdB new QPushButton("B");

QPushButton\* pcmdC new QPushButton 1 "С") ;

QPushButton\* pcmdD new QPushBut ton 1 "D") ;

QGridLayout\* pgrdLayout = new QGridLayout;//создали компоновку табличного размещения

pgrdLayout->setContentsMargins(5, 5, 5, 5);//отступ от границы

pgrdLayout->setSpacing(l5);//расст. Между виджетами

pgrdLayout->addWidget(pcmdA, О, 0);

pgrdLayout->addWidget(pcmdB, о, 1);

pgrdLayout->addWidget(pcmdC, 1, 0);

pgrdLayout->addWidget(pcmdD, 1, 1) ;

wgt.setLayout(pgrdLayout);

wgt.show();

return арр.ехес();

}

1. Стандартные диалоговые окна работы с файлами в Qt.
2. Стандартное диалоговое окно настройки принтера в Qt.
3. Стандартное диалоговое окно выбора цвета, выбора шрифта в Qt.
4. Стандартное диалоговое окно ввода, окно прогресса в Qt.
5. Три типа окон сообщений в Qt.
6. Воздушные подсказки, подсказка «Что это?» в Qt.
7. Система помощи – навигатор HelpBrowser в Qt .
8. Простой список – класс QListWidget.
9. Выбор элементов пользователем в списке QListWidget.
10. Режим пиктограмм, иерархические списки в Qt.
11. Таблицы, выпадающий список в Qt.
12. Операторы SQL: создание таблицы, добавление данных.
13. Операторы SQL: выборка, изменение данных, удаление данных.
14. Три уровня классов модуля QtSql.
15. Алгоритм соединения с базой данный в Qt, чтение данных из базы.
16. Класс QSqlQuery: запросы, навигация. 26. Способы построения запроса INSERT.
17. Концепция интервью в Qt: показ базы данных, выборка.
18. Характеристики и методы класса QPainter.
19. Класс QCanvas.
20. OpenGL в Qt.
21. Класс QDate : методы.
22. Класс QDate : создание своего формата времени, изменение времени.
23. Класс QTime: методы.
24. Класс QTime: вычисление времени работы функции.
25. Класс: QDateTime.
26. Класс QTime: подобие таймера.
27. Таймер, события таймера.
28. Применение таймера.
29. Одноразовый таймер, таймер с функцией connect().
30. Класс QBasicTimer.
31. Формат XML, его применение.
32. Структура XML-файла, теги и их особенность.
33. Модель DOM: иерархия документа.
34. Модель DOM: чтение XML-документа.
35. Модель DOM: создание и запись XML-документа.
36. Модель SAX: иерархия документа.
37. Модель SAX: чтение XML-документа.
38. Модель SAX: создание и запись XML-документа.
39. Основные классы модуля QtWebKit, свойства класса QWebView.
40. Методы класса QWebView, изменение настроек виджета WebView с помощью класса QWebSettings.
41. Класс QWebHistory, отображение истории работы с браузером.
42. Возможности класса QNetworkAccessManager, взаимодействие между QtNetWorkAccessManager и вэб-ресурсом.
43. Сигнал finished([QNetworkReply\*](http://doc.crossplatform.ru/qt/4.7.x/qnetworkreply.html) reply): вызов, параметры, получение ответа.
44. QtNetWorkAccessManager: Формирование POST и GET запросов.
45. Класс QCache, механизм кэширования.
46. Контекстно-независимое представление растровых изображений в Qt.
47. Контекстно-зависимое представление растровых изображений в Qt.
48. Использование каскадных стилей документа в Qt Creator.
49. Сохранение настроек приложения.
50. Область оповещений на рабочем столе.