Проект ООТРПО

Дайджест по третьему этапу

Участники проекта: гр. 5303 Допира В, Бочкарев И, Ильянов В., гр. 5304 Павлов Д

Выбранный контейнер: многодольный граф

В дайджесте представлены:

- План итерации.
- Демонстрация работы системы
- План тестирования
- Материалы поддержки пользователей
- План развёртывания

Система автоматизированной генерации расписания

План итераций

Версия 1.0

История Ревизий

Дата	**Версия**	**Описание**	**Автор**
22.05.2020	1.0	Первоначальная версия	Павлов Данила

1. Введение

1.1 Цель

Цель документа – описание плана итерации проекта. В данной итерации проводится реализация большей части функциональности продукта.

1.2 Определения и сокращения

Представлены в артефакте Глоссарий.

1.3 План

График сдачи каждой задачи проекта представлен ниже (см. табл.1).

Таблица 1 – График проекта

Фаза	***Задача***	***Окончание работы***	***Исполнители***
Построение	План итерации	22.05.2020	Павлов Данила
	Разработка системы	03.06.2020	Допира Валерия, Бочкарев Иван, Павлов Данила, Ильянов Вячеслав
	Тестирование системы	22.05.2020	Допира Валерия, Бочкарев Иван
	Материалы поддержки пользователей	22.05.2020	Ильянов Вячеслав
	План развёртывания	22.05.2020	Допира Валерия

2. Нагрузка исполнителей

- Допира Валерия

Таблица 2 – Нагрузка на исполнителя 1

Задача	***Окончание работы***	***Затраченное время***	***Исполнители***
Разработка системы	03.06.2020	7 дней	Допира Валерия, Бочкарев Иван, Павлов Данила, Ильянов Вячеслав
Тестирование системы	22.05.2020	2 дня	Допира Валерия, Бочкарев Иван
План развёртывания	22.05.2020	1 день	Допира Валерия

- Бочкарев Иван

Таблица 3 – Нагрузка на исполнителя 2

Задача	***Окончание работы***	***Затраченное время***	***Исполнители***
Разработка системы	03.06.2020	7 дней	Допира Валерия, Бочкарев Иван, Павлов Данила, Ильянов Вячеслав
Тестирование системы	22.05.2020	2 дня	Допира Валерия, Бочкарев Иван

- Павлов Данила

Таблица 4 – Нагрузка на исполнителя 3

Задача	***Окончание работы***	***Затраченное время***	***Исполнители***
План итерации	22.05.2020	1 день	Павлов Данила
Разработка системы	03.06.2020	7 дней	Допира Валерия, Бочкарев Иван, Павлов Данила, Ильянов Вячеслав

- Ильянов Вячеслав

Таблица 5 – Нагрузка на исполнителя 4

Задача	***Окончание работы***	***Затраченное время***	***Исполнители***
Разработка системы	03.06.2020	7 дней	Допира Валерия, Бочкарев Иван, Павлов Данила, Ильянов Вячеслав
Материалы поддержки пользователей	22.05.2020	2 дня	Ильянов Вячеслав

3. Диаграмма Ганта

Для иллюстрации плана, графика работ и занятости членов команды, работающих над проектом, удобно использовать диаграмму Ганта. Диаграмма Ганта со списком рабочих продуктов и исполнителями представлена ниже (см. рис.1).

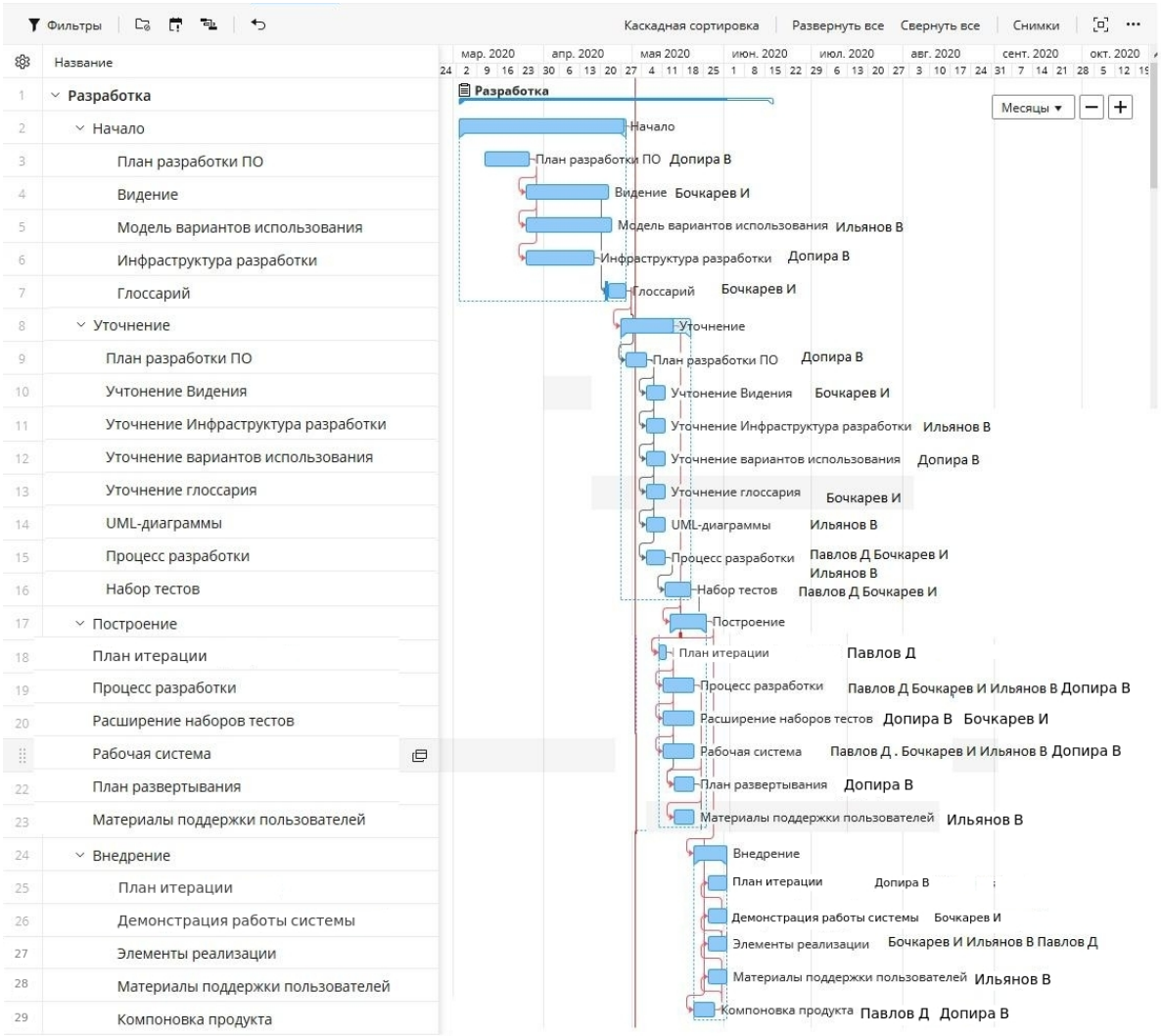


Рисунок 1 - Диаграмма Ганта

История Ревизий

Дата	Версия	Описание	Автор
29.05.2020	1.0	Первоначальная версия	Допира Валерия, Бочкарев Иван, Павлов Данила, Ильянов Вячеслав

Демонстрация системы

Ссылка на видеоматериалы:

https://drive.google.com/open?id=1_fy_S-CCY1uhef3Mi2IZ36LuDdoEmFs2

Описание:

На видео продемонстрирована текущая версия программы, содержащая ключевые компоненты (контейнер). Данные загружаются из файла формата JSON. Также через приложение можно создать новый файл, открыть, изменить и сохранить текущий (сохранить и сохранить как).

При нажатии на элементы таблиц и затем правую мышку открывается контекстное меню. В окне данных их можно добавлять, редактировать и удалять. В окне групп можно соотнести группу, предмет и указать количество часов. Все действия влияют на используемую модель (двудольный граф), добавляя и удаляя ребро между двумя долями. Расписание пока не выводится в приложении, будет добавлено позже. Визуализируется граф во вкладке «Визуализация графа». Доступно удаление вершин и фильтрация.

Класс граф , отвечающий за построение графа, имеет следующие связи с другими классами:

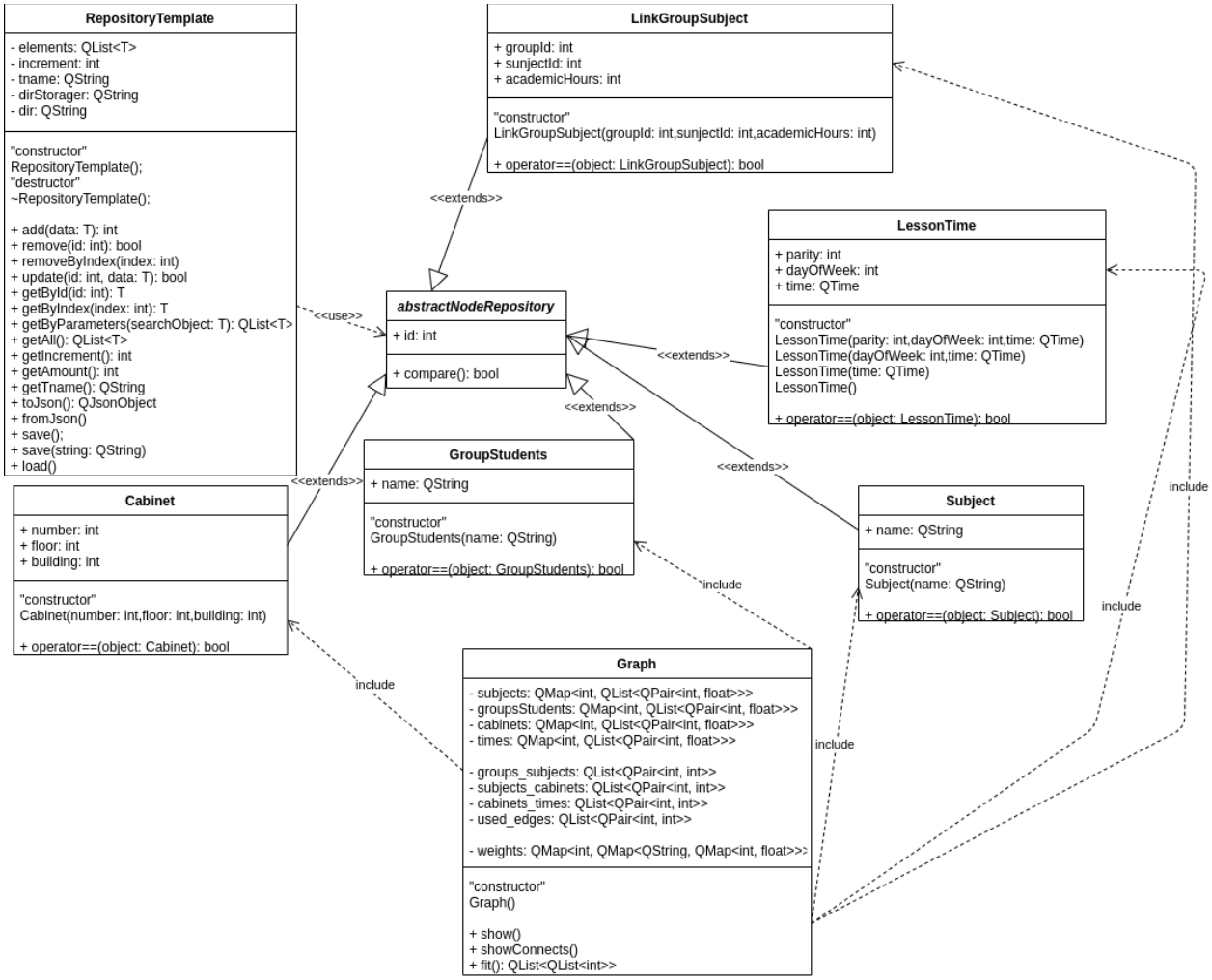


Рисунок 2 - Диаграмма

Алгоритм построения расписания описан ниже. Он подразумевает, что используется четырех дольный ориентированный граф.

Алгоритм генерации расписания:

– **Получение всех необходимых данных**. Данные берутся из репозиторий «кабинетов», «предметов», «групп», «времени» и «связей группы с предметами».

```
Graph::Graph(
    RepositoryTemplate<Cabinet> repoCabinets,
    RepositoryTemplate<GroupStudents> repoGroupStudents,
    RepositoryTemplate<LessonTime> repoLessonTime,
    RepositoryTemplate<Subject> repoSubjects,
    RepositoryTemplate<LinkGroupSubject> repoLinkGroupSubject
)
{
    this->repoCabinets = repoCabinets;
    this->repoGroupStudents = repoGroupStudents;
```

```
    this->repoLessonTime = repoLessonTime;
    this->repoSubjects = repoSubjects;
    this->repoLinkGroupSubject = repoLinkGroupSubject;
    //this->initGraph();
}
```

– **Инициализация графа**. В первую очередь создаются доли графа на основе репозиторев «предметов», «групп», «кабинетов» и «времени». На основе репозитория «связей группы с предметами» выстраиваются ребра между предметами и группами, при чем количество ребер между группой и предметом определяется количеством академических часов, выделанных группе на предмет. Все вершины доли «предметы» соединяются со всеми вершинами доли «кабинеты», а те в свою очередь со всеми вершинами доли «время».

```
void Graph::initGraph()
{
    for (int i=0; i < this->repoLinkGroupSubject.getAmount(); ++i) {
        LinkGroupSubject linkGroupSubject = this-
>repoLinkGroupSubject.getByIndex(i);
        QPair<int, int> item;
        item.first = linkGroupSubject.groupId;
        item.second = linkGroupSubject.subjectId;
        this->groups_subjects.append(item);
    }
    for (int i=0; i < this->repoSubjects.getAmount(); ++i) {
        Subject subject = this->repoSubjects.getByIndex(i);
        QList<QPair<int, float>> tmp = {};
        this->subjects.insert(subject.id, tmp);
    }
    for (int i=0; i < this->repoCabinets.getAmount(); ++i) {
        Cabinet cabinet = this->repoCabinets.getByIndex(i);
        QList<QPair<int, float>> tmp = {};
        this->cabinets.insert(cabinet.id, tmp);
    }
    for (int i=0; i < this->repoLessonTime.getAmount(); ++i) {
        LessonTime times = this->repoLessonTime.getByIndex(i);
        QList<QPair<int, float>> tmp = {};
        this->times.insert(times.id, tmp);
    }
}
```

– **Первичная генерация графа**. Для каждой вершины из доли «группы» случайным образом определяется один путь от «группы» до «времени», через «предмет» и «кабинет». Данный путь является корневым для «группы».

```
void Graph::firstStep()
{
    // Проходимся по всем группам учащихся
    QList<int> keys = this->groupsStudents.keys();
    for (int key : keys) {
        srand(time(0));
        // Получаем объект группы студентов
        QList<QPair<int, float>> groupStudents = *this-
>groupsStudents.find(key);
```

```
            // Выбираем случайный предмет
            int r = rand() % groupStudents.size();
            QPair<int, float> subject = groupStudents[r];
            // Выбираем случайный кабинет
            r = rand() % this->repoCabinets.getAmount();
            auto cabinet = this->repoCabinets.getByIndex(r);
            // Закрепляем за предметом кабинет
            QPair<int, float> item;
            item.first = cabinet.id;
            item.second = 0;
            QList<QPair<int, float>> tmp = { item };
            this->subjects.remove(subject.first);
            this->subjects.insert(subject.first, tmp);
            // Чтобы не было повторений
            while (true) {
                srand(time(0));
                int r = rand() % this->repoLessonTime.getAmount();
                auto time = this->repoLessonTime.getByIndex(r);
                auto cabtime = this->cabinets[cabinet.id];
                QPair<int, float> item;
                item.first = time.id;
                item.second = 0;
                if(cabtime.count(item) == 0) {
                    tmp = { item };
                    this->cabinets.remove(cabinet.id);
                    this->cabinets.insert(cabinet.id, tmp);
                    break;
                }
            };
        }
}
```

– **Инициализация весов**. Основываясь на корневом пути для каждой группы инициализируются веса ребер, таким образом, что веса от доли «группа» к доле «предметы» равны нулю, веса от доли «предметы» к доле «кабинеты» определяются по формуле:

$$\sqrt{\left(корпус_i - корпу_{корневой}c\right)^2 + \left(этаж_i - этаж_{корневой}\right)^2 + \left(кабинет_i - кабинет_{корневой}\right)^2} = вес$$

Данная формула помогает рассчитать подобие географического расстояния между аудиториями.

Веса для ребер между кабинетом и временем рассчитываются похожим способом, за тем лишь исключением, время предварительно нормируется:

$$\sqrt{\left(день_{неделиi} - день_{неделикорневой}c\right)^2 + \left(четность_i - четность_{корневой}\right)^2 + \left(время_i - время_{корневой}\right)^2} = вес$$

```
QList<QList<int>> Graph::fit()
{
    RepositoryTemplate<Cabinet> cabinets = this->repoCabinets;
    RepositoryTemplate<LessonTime> times = this->repoLessonTime;
```

```
    QMap<int, QList<QPair<int, int>>> ways;
    QList<QPair<int, int>> used_edges_local;
    QList<int> used_times;
    QList<QPair<int, int>> edges;
    QPair<int, int> edge;
    QList<QPair<int, float>> cabinets_w;
    QList<QPair<int, float>> times_w;
    QPair<int, float> pair;
    int k;
    int r;
    for (Cabinet cab : cabinets.getAll()) {
        edge.first = cab.id;
        for (LessonTime time : times.getAll()) {
            edge.second = time.id;
            edges.append(edge);
        }
    }
    for (GroupStudents group : this->repoGroupStudents) {
        cabinets_w.clear();
        times_w.clear();
        used_times.clear();
        used_edges_local.clear();
        auto links_subjects = this-
>repoLinkGroupSubject.getByParameters(LinkGroupSubject(group.id,-1,-1));
        srand(group.id+time(0));
        r = rand() % links_subjects.size();
        LinkGroupSubject start_link = links_subjects[r];
        r = rand() % cabinets.getAmount();
        Cabinet start_cabinet = cabinets.getByIndex(r);
        r = rand() % times.getAmount();
        LessonTime start_time = times.getByIndex(r);
        qSort(edges.begin(), edges.end(), [this, start_cabinet, start_time]
(QPair<int, int>& a, QPair<int, int>& b) {
            Cabinet cab1 = this->repoCabinets.getById(a.first);
            Cabinet cab2 = this->repoCabinets.getById(b.first);
            LessonTime time1 = this->repoLessonTime.getById(a.second);
            LessonTime time2 = this->repoLessonTime.getById(b.second);
            float c1 = sqrt(pow(cab1.number - start_cabinet.number, 2) +
pow(cab1.floor - start_cabinet.floor, 2) + pow(cab1.building -
start_cabinet.building, 2));
            float c2 = sqrt(pow(cab2.number - start_cabinet.number, 2) +
pow(cab2.floor - start_cabinet.floor, 2) + pow(cab2.building -
start_cabinet.building, 2));
            float t1 = sqrt(pow(time1.dayOfWeek - start_time.dayOfWeek,2) +
pow(time1.parity - start_time.parity,2) +
pow((start_time.time.msecsSinceStartOfDay()/8640000+start_time.dayOfWeek*100) -
(time1.time.msecsSinceStartOfDay()/8640000+time1.dayOfWeek*100),2));
            float t2 = sqrt(pow(time2.dayOfWeek - start_time.dayOfWeek,2) +
pow(time2.parity - start_time.parity,2) +
pow((start_time.time.msecsSinceStartOfDay()/8640000+start_time.dayOfWeek*100) -
(time2.time.msecsSinceStartOfDay()/8640000+time2.dayOfWeek*100),2));
            return c1 + t1 < c2 + t2;
        });
        for (LinkGroupSubject link : links_subjects) {
            for (int i = 0, amount = link.academicHours; i < amount; ++i) {
                for (k = 0; used_times.count(edges[k].second) != 0; ++k);
                pair = edges[k];
                edges.removeAt(k);
                used_times.append(pair.second);
                used_edges_local.append(pair);
```

```
            }
        }
        ways.insert(group.id, used_edges_local);
    }
    QList<QList<int>> result;
    QList<int>way;
    for (LinkGroupSubject link : this->repoLinkGroupSubject) {
       for (int i = 0; i < link.academicHours; ++i) {
            way.clear();
            pair = ways[link.groupId].front();
            ways[link.groupId].pop_front();
            way.append(link.groupId);
            way.append(link.subjectId);
            way.append(pair.first);
            way.append(pair.second);
            result.append(way);
            ++k;
        }
    }
    return result;
}
```

— **Поиск минимального пути (генерация расписания).** Все ребра сортируются по возрастанию весов. После чего запускается цикл. На каждой итерации выбирается ребро между двумя соединенными долями с минимальным весом и помечается как использованное. Таким образом выбранные ребра становятся ребрами нового пути.

```
void Graph::showConnects() {
    QMapIterator<int, QList<QPair<int, float>>> it(this->groupsStudents);
    while (it.hasNext()) {
        it.next();
        auto subjects = it.value();
        for (auto subject : subjects) {
            auto cabinets = this->subjects[subject.first];
            for (auto cabinet : cabinets) {
                auto times = this->cabinets[cabinet.first];
                for (auto time : times) {
                    GroupStudents g = this->repoGroupStudents.getById(it.key());
                    Subject s = this->repoSubjects.getById(subject.first);
                    Cabinet c = this->repoCabinets.getById(cabinet.first);
                    LessonTime t = this->repoLessonTime.getById(time.first);
                    qDebug() << g.toString().toLocal8Bit().data()  << " -> " <<
s.toString().toLocal8Bit().data() << " -> " << c.toString().toLocal8Bit().data()
<< " -> "  << t.toString().toLocal8Bit().data() << endl;
                }
            };
        }
    }
}
```

— **Постобработка данных.** Перед выдачей результата формируется итоговый список ребер, который и передается дальше.

```
void Graph::show()
{
    for (int i=0; i < this->repoCabinets.getAmount(); ++i) {
```

```
        Cabinet cabinet = this->repoCabinets.getByIndex(i);
        qDebug() << cabinet.toString() << endl;
    }
    qDebug() << this->repoCabinets.getAmount() << endl;
    for (int i=0; i < this->repoSubjects.getAmount(); ++i) {
        Subject subject = this->repoSubjects.getByIndex(i);
        qDebug() << subject.toString() << endl;
    }
    qDebug() << this->repoSubjects.getAmount() << endl;
    for (int i=0; i < this->repoLessonTime.getAmount(); ++i) {
        LessonTime lessonTime = this->repoLessonTime.getByIndex(i);
        qDebug() << lessonTime.toString() << endl;
    }
    qDebug() << this->repoLessonTime.getAmount() << endl;
    for (int i=0; i < this->repoGroupStudents.getAmount(); ++i) {
        GroupStudents groupStudents = this->repoGroupStudents.getByIndex(i);
        qDebug() << groupStudents.toString() << endl;
    }
    qDebug() << this->repoGroupStudents.getAmount() << endl;
    for (int i=0; i < this->repoLinkGroupSubject.getAmount(); ++i) {
        LinkGroupSubject linkGroupSubject = this-
>repoLinkGroupSubject.getByIndex(i);
        qDebug() << linkGroupSubject.toString() << endl;
    }
    qDebug() << this->repoLinkGroupSubject.getAmount() << endl;
    qDebug() << "groups-subjects" <<this->groups_subjects;
}
```

Система автоматизированной генерации расписания

Тестирование системы

Версия 1.0

История Ревизий

Дата	Версия	Описание	Автор
22.05.2020	1.0	Первоначальная версия	Допира Валерия, Бочкарев Иван

1. Введение

1.1. Цель

Целью данного плана тестирования является описание тестирования разработанного приложения.

1.2. Контекст

Целью тестирования является тщательная проверка всех функций приложения. Будут протестированы как позитивные, так и негативные сценарии, которые также могут выявить некоторые ошибки.

Ручное тестирование уже было проведено во 2 итерации.

1.3. Определения, акронимы, сокращения

См. глоссарий проекта.

2. Стратегии тестирования

В результате первого цикла тестирования, где проводятся функциональные тесты, будут внесены некоторые исправления и дополнения и внесены в план тестирования. Первый цикл даст определенное понимание стабильности системы и поможет определить необходимый набор тестов, который будет выполнен в процессе тестирования приложения. Такой метод даст возможность получить подробный отчет о продукте.

Планируется четыре этапа тестирования:

– первый этап – анализ, создание плана тестирования, частичное выполнение некоторых функциональных тестов;

– второй этап будет посвящен подробному выполнению функциональных тестов, раскрывающих и описывающих ошибки;

– третий этап – проверка исправленных ошибок и выполнение регрессионного тестирования;

– четвёртый этап заключается в тестировании пользовательского интерфейса с раскрытием и описанием ошибок.

Такая система тестирования позволяет выполнять детальное тестирование и предотвращать, исправлять ошибки на ранних этапах.

2.1. Виды тестирования

2.1.1. Функциональное тестирование

Цель: обнаружение функциональных ошибок посредством выполнения функциональных тестов.

2.1.2. Регрессионное тестирование

Цель: проверка изменений, внесенных в приложение, чтобы убедиться, что в новой версии нет ошибок в тех частях, где тестирование уже было выполнено.

2.1.3. Тестирование удобства пользования

Цель: дать оценку уровня удобства использования приложения по следующим пунктам:

– производительность, эффективность – количество времени и/или шагов необходимое пользователю для завершения основных задач приложения;

– правильность – количество ошибок, сделанное пользователем во время работы с приложением;

– активизация в памяти – как много пользователь помнит о работе приложения после приостановки работы с ним на длительный период времени;

– эмоциональная реакция – ощущения пользователя после завершения задачи.

2.1.4. Тестирование производительности

Цель: определение масштабируемости приложения под нагрузкой, при этом происходит:

– измерение времени выполнения выбранных операций при определенных

интенсивностях выполнения этих операций;

– определение количества пользователей, одновременно работающих с приложением.

2.1.5 Тестирование стабильности (надежности)

Цель: проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль.

2.1.6 Тестирование на отказ и восстановление

Цель: проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта. Объектом тестирования являются весьма вероятные эксплуатационные проблемы, такие как:

– отказ электричества на компьютере-сервере;

– отказ электричества на компьютере-клиенте;

– незавершенные циклы обработки данных (прерывание работы фильтров данных, прерывание синхронизации);

– объявление или внесение в массивы данных невозможных или ошибочных элементов;

– отказ носителей данных.

3. ТЕСТИРОВАНИЕ

Приложение было протестировано, внесены изменения.

3.1. Функциональное тестирование

3.1.1 Визуализация графа в классе `GraphWidget`

Задается сцена и ее параметры. Затем на нее помещаются 5 точек разных цветов (класс `Node`). Позиции устанавливаются с помощью функции `setPos()`. Некоторые из них соединяются ребрами с помощью функции `addItem()` и класса `Edge`. Код:

```
scene->setItemIndexMethod(QGraphicsScene::NoIndex);
scene->setSceneRect(-200, -200, 400, 400);
```

```
    setScene(scene);
    setCacheMode(CacheBackground);
    setViewportUpdateMode(BoundingRectViewportUpdate); //reconsider
    setRenderHint(QPainter::Antialiasing);
    setTransformationAnchor(AnchorUnderMouse);
    scale(qreal(1), qreal(1));
    //setMinimumSize(400, 400);

    //@brief
    Node *node1 = new Node (this,15,Qt::cyan);
    Node *node2 = new Node (this,15,Qt::gray);
    Node *node3 = new Node (this,15,Qt::red);
    Node *node4 = new Node (this,15,Qt::blue);
    Node *node5 = new Node (this,15,Qt::green);


    scene->addItem(node1);
    scene->addItem(node2);
    scene->addItem(node3);
    scene->addItem(node4);
    scene->addItem(node5);

    scene->addItem(new Edge(node1, node2));
    scene->addItem(new Edge(node2, node3));
    scene->addItem(new Edge(node2, node4));
    scene->addItem(new Edge(node2, node5));
    scene->addItem(new Edge(node5, node4));

    node1->setPos(0,0);
    node2->setPos(0,60);
    node3->setPos(20,10);
    node4->setPos(40,25);
    node5->setPos(10,-30);

    nodes.push_back(node1);
    nodes.push_back(node2);
    nodes.push_back(node3);
    nodes.push_back(node4);
    nodes.push_back(node5);

    readGraph(QPointF(-30,-20));
```

Результат представлен на рисунке 2. Ожидаемый результат соответствует полученному.

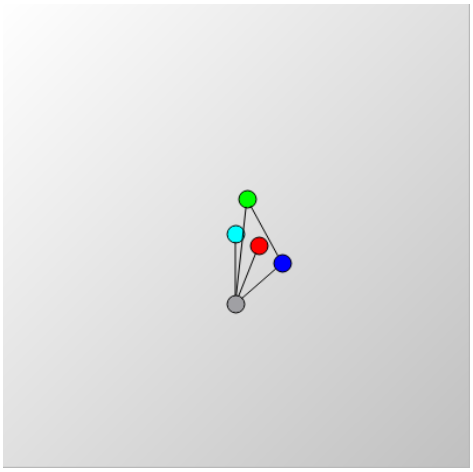


Рисунок 2 — Визуализация графа

3.1.2 Проверка на наследование `Node` от `abstractNodeRepository` в классе `RepositoryTemplate`

```
template <class T>
void RepositoryTemplate<T>::checkNode() {
   abstractNodeRepository* test = dynamic_cast<abstractNodeRepository*> (new T);
   if (!test) {
        throw std::runtime_error("The type of template does not satisfy the
required. The type must be a descendant of the abstractNodeRepository class.");
   }
}
```

Ожидаемый результат соответствует полученному. Ошибок не было.

3.1.3 Проверка на изменения (удаление, добавление, редактирование) репозиториев в главной вкладке в классе `DialogLinkGroupSubjectWindow`

Для группы:

```
void
DialogLinkGroupSubjectWindow::editDataRepoGroup(RepositoryTemplate<GroupStudents
> repoGroupStudents){
          //Проверка на изменения(удаление, добавление, редактирование)
репозиториев в главной вкладке
    if (repoRecGroupStudent.getAmount()==0){
        for (int i =0; i<repoGroupStudents.getAmount(); i++){
            repoRecGroupStudent.add(repoGroupStudents.getByIndex(i));
        }

    }else
          if (repoGroupStudents.getAmount()>repoRecGroupStudent.getAmount()){
                                    int   raz   =   repoGroupStudents.getAmount()-
repoRecGroupStudent.getAmount();
            int addE = repoRecGroupStudent.getAmount();
            for (int i =0; i<raz; i++){
                repoRecGroupStudent.add(repoGroupStudents.getByIndex(i));
                ++addE;
            }
    }else
```

```
          if (repoGroupStudents.getAmount()<repoRecGroupStudent.getAmount()){
                                  //int   raz   =   repoRecGroupStudent.getAmount()-
repoGroupStudents.getAmount();
            //int delE = repoRecGroupStudent.getAmount()-1;
            for (int i =0; i<dinGr.size(); i++){
                repoRecGroupStudent.remove(repoRecGroupStudent.getById(repoRecGr
oupStudent.getByIndex(dinGr[i]).id).id);
            //--delE;
            }
    }
    else
         if (repoGroupStudents.getAmount()==repoRecGroupStudent.getAmount()){
            for (int i =0; i<repoGroupStudents.getAmount(); i++){
                                                                                if
(repoGroupStudents.getById(repoGroupStudents.getByIndex(i).id).name!
=repoRecGroupStudent.getById(repoRecGroupStudent.getByIndex(i).id).name){
                     repoRecGroupStudent.update(repoRecGroupStudent.getById(repoR
ecGroupStudent.getByIndex(i).id).id,repoGroupStudents.getById(repoGroupStudents.
getByIndex(i).id).name);
                }
            }
          }
}
```

Ожидаемый результат соответствует полученному. Ошибок нет.

Для предмета:

```
void
DialogLinkGroupSubjectWindow::editDataRepoSubject(RepositoryTemplate<Subject>
repoSubjects){
          //Проверка на изменения(удаление, добавление, редактирование)
репозиториев в главной вкладке
    if (repoRecSubject.getAmount()==0){
        for (int i =0; i<repoSubjects.getAmount(); i++){
            repoRecSubject.add(repoSubjects.getByIndex(i));
        }

    }else
          if (repoSubjects.getAmount()>repoRecSubject.getAmount()){
            int raz = repoSubjects.getAmount()-repoRecSubject.getAmount();
            int addE = repoRecSubject.getAmount();
            for (int i =0; i<raz; i++){
                repoRecSubject.add(repoSubjects.getByIndex(addE));
                ++addE;
            }
    }else
          if (repoSubjects.getAmount()<repoRecSubject.getAmount()){
            //int raz = repoRecSubject.getAmount()-repoSubjects.getAmount();
            //int delE = repoRecSubject.getAmount()-1;
            for (int i =0; i<dinSb.size(); i++){
                repoRecSubject.remove(repoRecSubject.getById(repoRecSubject.getB
yIndex(dinSb[i]).id).id);
            //--delE;
            }
    }
    else
         if (repoSubjects.getAmount()==repoRecSubject.getAmount()){
            for (int i =0; i<repoSubjects.getAmount(); i++){
                  if (repoSubjects.getById(repoSubjects.getByIndex(i).id).name!
=repoRecSubject.getById(repoRecSubject.getByIndex(i).id).name){
```

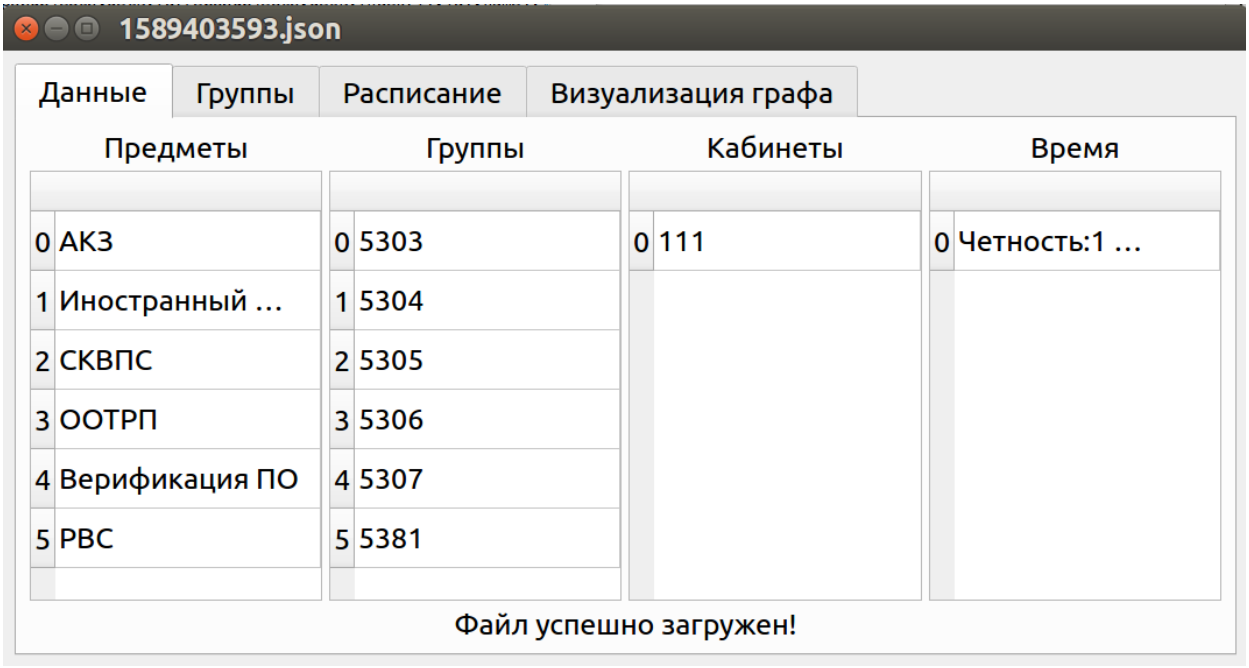
```
                    repoRecSubject.update(repoRecSubject.getById(repoRecSubject.
getByIndex(i).id).id,repoSubjects.getById(repoSubjects.getByIndex(i).id).name);
                }
            }
        }
    }
```

Ожидаемый результат соответствует полученному. Ошибок нет.

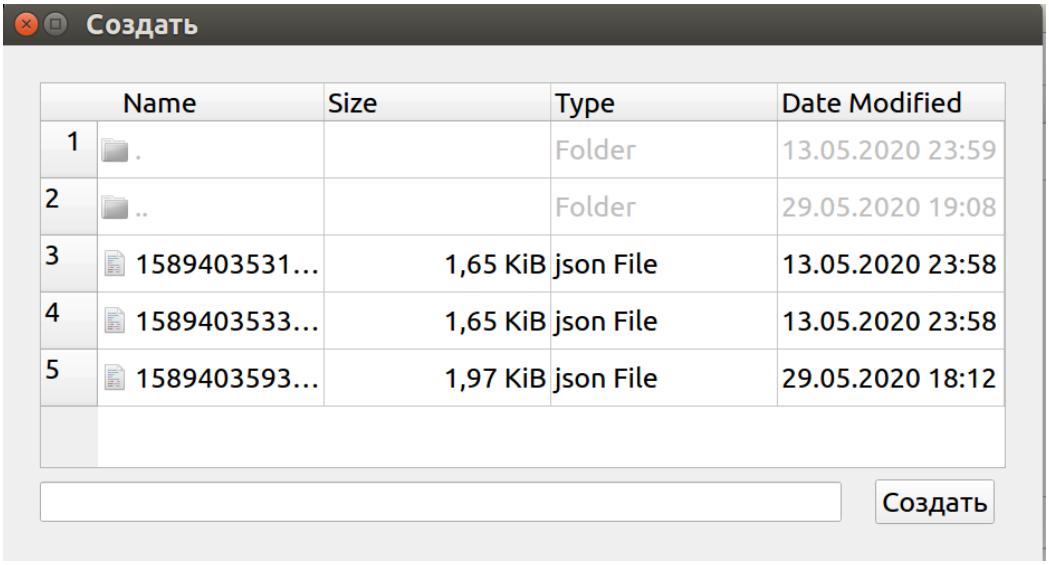
3.2. Регрессионное тестирование

Ручное тестирование описано во 2 итерации. После этого были реализованы следующие функции:

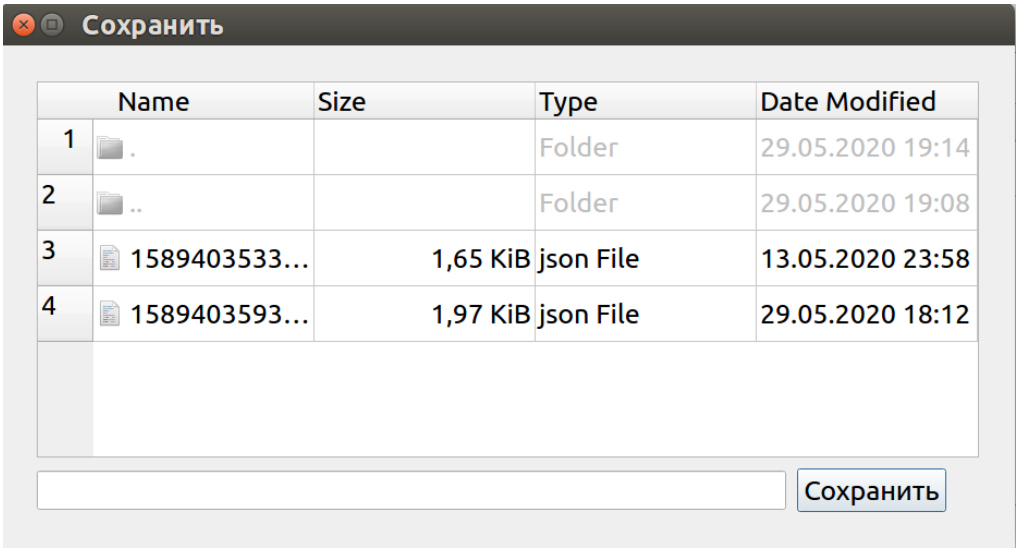
Открытие файла JSON при загрузке приложения и вывод сообщения:



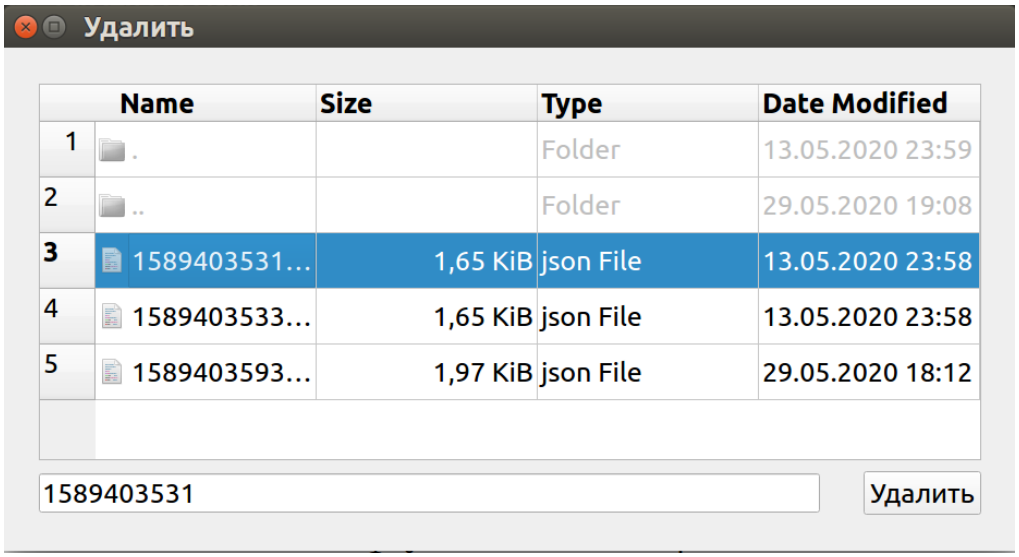
При создании открывается следующее окно:



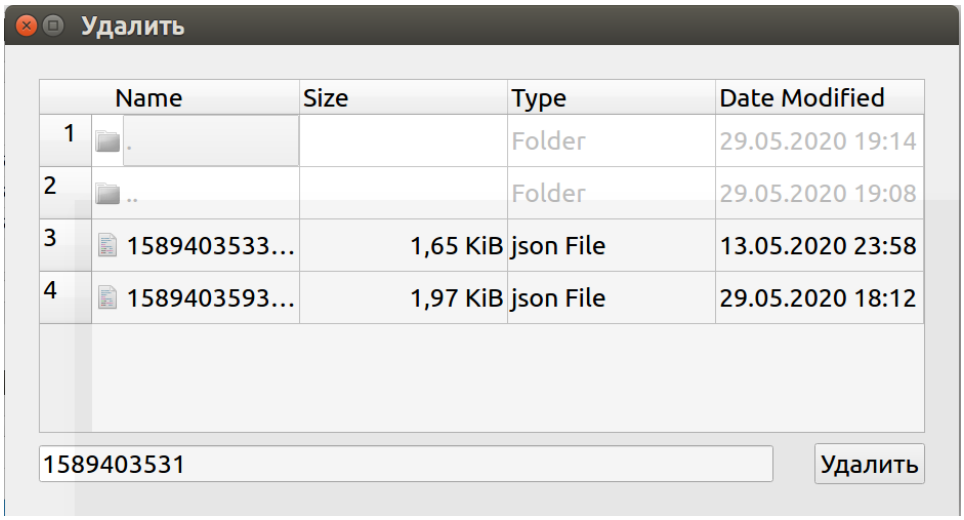
Сохранить как:



Удаление:



После удаления:



3.3. Тестирование удобства пользования

Если данные уже загружены и сохранены в JSON, то пользователю необходимо лишь перейти на вкладку Расписание для отображения расписания или на вкладку Визуализация для отображения графа. Для работы с файлом необходимо обратиться к меню приложения. Все действия перечислены подряд без вложенности, и при этом их немного, поэтому пользователь сможет легко найти необходимое. При работе с данными пользователю нужно открыть контекстное меню, ввести данные и нажать окей. Все действия занимают до 3-х шагов, поэтому можно сказать, что приложение имеет высокую эффективность.

Работа с данными осуществляется через контекстное меню, которое отображается при нажатии на правую мышку. Данный способ возможно не будет интуитивным для всех пользователей, но взаимодействие с приложением будет описано в документации, поэтому у пользователя не будет ошибок при работе с приложением.

Наличие документации обеспечивает воспроизводимость инструкций для работы с приложением, поэтому даже если пользователь не помнит, как работать из-за приостановки работы с ним на длительный период, пользователь всегда будет иметь указания.

Приложение направлено на осуществление определенной задачи — построение расписание. И если она будет выполнена, то пользователь будет удовлетворен. Он также сможет редактировать полученный результат, что тоже скажется на удобстве приложения.

3.4. Тестирование производительности

Приложение зависит от количества данных, которое ему необходимо будет обработать. Однако так как приложение рассчитано для работы со школой или факультетом, то оно сможет не заметно для пользователя обработать данные, которые обычно имеют такие организации.

Одновременно с приложением может работать 1 пользователь.

3.5 Тестирование стабильности (надежности)

Стабильность системы не зависит от времени работы приложения.

3.6 Тестирование на отказ и восстановление

При отключении электричества, выключении компьютера приложение будет завершено, а не сохраненные данные потеряны. Поэтому пользователю необходимо позаботиться о резервных источниках питания, периодически сохранять файл с данными.

Также добавлены проверки на ввод некоторых данных. Например, время проведения занятий. В некоторых местах важно, чтобы данные были только числовыми, например, количество академических часов. Дни недели пользователь выбирает из предложенного списка: Понедельник, вторник, среда, четверг, пятница, суббота, воскресенье. Предметы, названия групп могут иметь как числовые, так и буквенные значения.

4. Вывод

Конечным результатом является протестированное приложение без ошибок.

Система автоматизированной генерации расписания

Материалы поддержки пользователей

Версия 1.0

История Ревизий

Дата	Версия	Описание	Автор
22.05.2020	1.0	Первоначальная версия	Ильянов Вячеслав

1. Введение

1.1 Цель

Техническая поддержка служит для помощи конкретным пользователям решать возникающие конкретные проблемы с продуктом и его использованием, нежели задачи, связанные с обучением, индивидуальной настройкой или другими услугами поддержки.

2. Методы поддержки

Инструкция работы с приложением описано в документации и позволит решить многие задачи.

Пользователь сможет связаться как напрямую с разработками приложения, так и воспользоваться краудсорсингом. На гитхабе, где выложено приложение, будет возможность пообщаться с другими пользователями и найти способы решения проблемы.

Система автоматизированной генерации расписания

План развёртывания

Версия 1.0

История Ревизий

Дата	**Версия**	**Описание**	**Автор**
22.05.2020	1.0	Первоначальная версия	Допира Валерия

1. Введение

1.1 Цель

Развёртывание программного обеспечения (Развёртывание ПО, Software deployment) — это все действия, которые делают программную систему готовой к использованию. Данный процесс является частью жизненного цикла программного обеспечения.

В целом процесс развертывания состоит из нескольких взаимосвязанных действий с возможными переходами между ними. Эта активность может происходить как со стороны производителя, так и со стороны потребителя. Поскольку каждая программная система является уникальной, трудно предсказать все процессы и процедуры во время развертывания. Поэтому «развертывание» можно трактовать как общий процесс, соответствующий определенным требованиям и характеристикам. Развертывание может осуществляться программистом и в процессе разработки программного обеспечения.

2. Определение стратегий обеспечения совместимости, преобразования и переноса

Если системе предстоит заменить существующую систему, то необходимо учесть аспекты, связанные с совместимостью, преобразованием и переносом. Конкретно:

- Данные из существующей системы должны быть перенесены (и, возможно, преобразованы в соответствующий формат) в новую систему.

- Существующие пользовательские интерфейсы (экранные форматы, команды и т.п.) должны поддерживаться в новой системе.
- Должны поддерживаться все существующие интерфейсы прикладных программ (API).
- Перенос из существующей системы в новую не должен нарушить работу пользователей более чем на заранее определенный период (конкретное значение зависит от вида бизнеса).
- У новой системы должна быть возможность работать параллельно со старой системой в течение переходного периода.
- Должна существовать возможность переключения обратно на старую систему, если это понадобится, в течение первых двух недель работы.
- Старые архивные данные могут понадобиться в новой системе. Если они зашифрованы, то шифровальные ключи потребуют особого внимания во время переноса.

Стратегии, выбранные для решения перечисленных вопросов, потребуют соответствующей поддержки в архитектуре и конфигурации системы

3. Определение расписания развертывания

Перенос системы в рабочую среду требует планирования и подготовки. Ниже перечислены технические факторы, которые нужно учесть:

- Может потребоваться подготовка пользователей системы.
- Необходимо подготовить рабочую среду и обучить персонал, чтобы он мог поддерживать работу системы.
- Необходимо разработать процедуры поддержки рабочего процесса, в том числе процедуры резервного копирования, восстановления и устранения неполадок.

3.1 Бизнес - факторы

Далее рассмотрены бизнес-факторы, влияющие на расписание развертывания:

- По причинам, связанным с бизнесом, систему может потребоваться развернуть к конкретной дате; несоблюдение этого условия может

значительно снизить ценность системы.

- Могут быть периоды, когда развертывание системы невозможно по причинам, связанным с бизнесом или текущим рабочим процессом, включая, но не ограничиваясь этим, окончание отчетного финансового периода или период, в течение которого работа системы не должна прерываться.

Пики рабочей нагрузки и другие факторы в существующих системах и процессах могут препятствовать развертыванию в определенные моменты времени. Например:

1. Увеличенная нагрузка на систему: еженедельные, ежемесячные или ежегодные пики
2. Регулярные циклы обслуживания аппаратного или программного обеспечения - влияют и на готовность систем, и на персонал
3. Периоды выходных и отпусков
4. Плановые одноразовые нарушения рабочего процесса из-за модернизации аппаратного обеспечения или внедрения новых систем
5. Плановые реорганизации
6. Изменения устройств и средств.

- Некоторые системы должны работать постоянно (например, сетевые и телефонные коммутаторы); для таких систем развертывать новую версию нужно во время работы старой.

3.2 Определение последовательности развертывания

Некоторые системы должны развертываться постепенно, частями, по причинам, связанным с расписанием или готовностью. Если систему нельзя развернуть сразу целиком, то необходимо определить порядок установки компонентов и узлы, на которых они должны устанавливаться. Ниже перечислены общепринятые шаблоны планирования развертывания:

1. Географически - в зависимости от территории
2. Функционально - в зависимости от приложения
3. Организационно - в зависимости от подразделения или должностных

обязанностей

Во время поэтапного развертывания приложения необходимо учитывать следующие аспекты:

- программное обеспечение должно работать и в условиях неполной конфигурации
- различные версии программного обеспечения должны быть совместимы
- должна существовать возможность возврата к предыдущей версии системы в случае обнаружения неполадок в новой системе

3.3 Определение необходимости обучения пользователей

Для каждой категории пользователей - администраторов, операторов и обычных пользователей - определите следующее:

- С какими системами информационных технологий они имеют дело в настоящий момент? Если при работе с данной системой какие-либо категории пользователей, как внутренних, так и внешних по отношению к организации, впервые столкнутся с той или иной информационной технологией, то отметьте это как пункт, требующий особого внимания.
- С какими новыми функциями они встретятся в данной системе?
- Чему потребуется их обучить, в широком смысле этого слова?
- Каковы требования относительно поддержки национальных языков (NLS)?