Описание процессов, обеспечивающих поддержание жизненного цикла

ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

«DiaLog»

2023 г.

Введение

Под программным обеспечением (далее – «**ПО**») в настоящем документе понимается программное обеспечение «**DiaLog**», правообладателем которого является общество с ограниченной ответственностью «МИКС» (далее – «МИКС**»**).

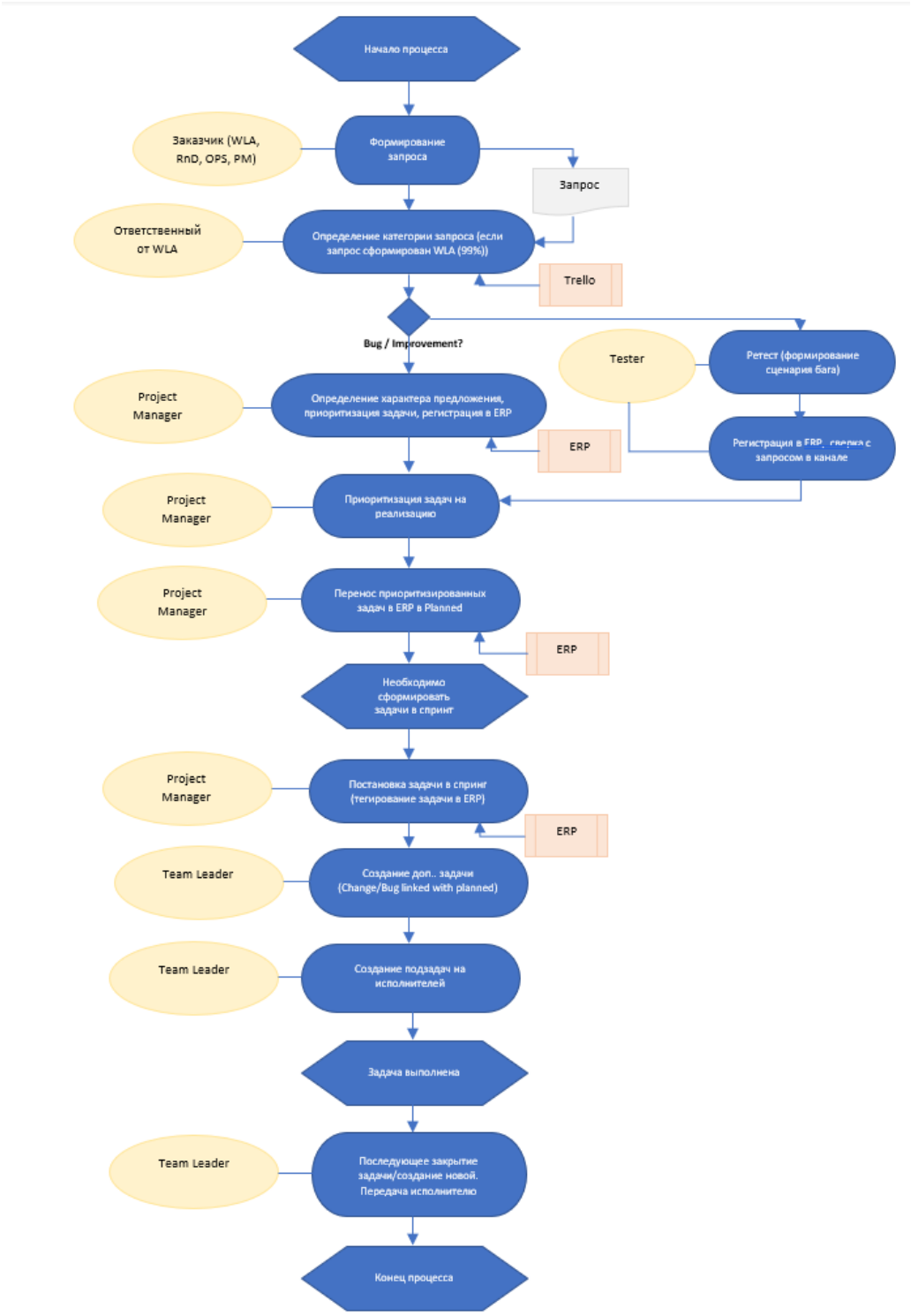
Настоящий документ подготовлен в целях предоставления информации для формирования заявления о включении сведений о ПО в Единый реестр российских программ для ЭВМ и баз данных[1].

Оглавление

[1] Указанную информацию необходимо предоставлять с 01 января 2023 года в связи с принятием Постановления Правительства РФ от 28 декабря 2022 № 2461 «О внесении изменений в постановление Правительства Российской Федерации от 16 ноября 2015 № 1236.

1. Процессы реализации (разработки) ПО «DiaLog»

Общий процесс взаимодействия сотрудников Отдела программного обеспечения в процессе реализации представлен на схеме:



1.1. Проектирование

Проектирование — процесс разработки проекта, от этапа получения запроса на реализацию, его анализа и подготовки технического задания до тестирования продукта и приемки его заказчиком.

Координатор проекта принимает решение о реализации/доработке изменения в соответствии с информацией, полученной на стадии Анализа и Оценки.

Если выполняется хотя бы одно условие:

- трудозатраты на реализацию изменения больше 400 ч.ч.;
- существующие функциональные или нефункциональные возможности в результате изменения станут полностью или частично не доступны;
- реализация изменения отменяет или искажает другие изменения или влияет на сроки реализации других изменений более чем на 7 календарных дней;

то Координатор проекта организует консультативный совет по изменениям для принятия решения о реализации изменения, иначе Координатор проекта сам принимает решение о реализации (изменение будет реализовано или нет), основываясь на результатах анализа о технической возможности реализации.

1.2. Конструирование

Конструирование продукта осуществляется разработчиками Отдела программного обеспечения, согласно нижеприведенным принципам и правилам разработки:

- Для решения задач при проектировании рассматривается несколько решений.
- В проекте должно быть четко определено, какие параметры конфигурируются, а какие описываются в константах или вычисляются из окружения среды.
- Код должен быть дружественен к 64- и 32-битных системам.
- Статические переменные типа «класс» рекомендуется объявлять как constexpr (для C++14).
- Статические переменные необходимо объявлять в локальном пространстве имен.
- «Магические числа» и строковые значения в коде описываются в константах или переменных.

Пример (из wiki):
такой фрагмент будет плохим:

```
drawSprite( 53, 320, 240 );
```

его следует реализовать:

```
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_X_CENTER = SCREEN_WIDTH/2;
const int SCREEN_Y_CENTER = SCREEN_HEIGHT/2;
const int SPRITE_CROSSHAIR = 53;
...
```

```
drawSprite( SPRITE_CROSSHAIR, SCREEN_X_CENTER, SCREEN_Y_CENTER );
```

- При работе с ресурсам следует использовать идиому RAII.

Пример:

Плохой код:

```
File f;
f.open("boo.txt");
//UNSAFE - an exception here means the file is never closed
loadFromFile(f);
f.close();

Dog* dog = new Daschund;
//UNSAFE - an exception here means the dog is never deleted
goToThePark(dog);
delete dog;

Lock* lock = getLock();
lock.aquire();
//UNSAFE - an exception here means the lock is never released
doSomething();
lock.release();
```

Хороший код:

```
class OpenFile
{
public:
    OpenFile(const char* filename)
    {
        //throws an exception on failure (исключение для пункта «6. Конструкторы не должны выполнять
действия
        //вызывающие необрабатываемые исключения»)
        file.open(filename);
    }

    ~OpenFile(){
        _file.close();
    }

    std::string readLine() {
        return _file.readLine();
    }

private:
    File _file;
};

// then we can use it like this
OpenFile f("boo.txt");
//exception safe, and no closing necessary
loadFromFile(f);
```

- Не следует создавать RAII объекты в heap:

```
std::string firstLineOf(const char* filename)
{
    OpenFile f("boo.txt"); //stack allocated
    return f.readLine();
    //File closed here. `f` goes out of scope and destructor is run.
}

std::string firstLineOf(const char* filename)
{
    OpenFile* f = new OpenFile("boo.txt"); //heap allocated
    return f->readLine();
    //Destructor is never run, because `f` is never deleted
}
```

- Наследование используется только для типа отношения «является», в остальных случаях используется Композиция.

Наследование:

```
class Point
{
    int x, y;
};

class Circle: public Point
{

};
```

Композиция:

```
class Point
{
    int x, y;
};

class Circle
{
    Point p1;
};
```

Или менее жесткая форма – Агрегация:

```
class Point
{
    int x, y;
};

class Circle
{
    Point *p1;

public:
    Circle( Point *Center )
      :p1(Center)
    { }
};
```

- Конструкторы классов не должны вызывать виртуальные методы и не должны выполнять инициализацию, в результате которой может возникнуть исключение (если не предусмотрена обработка исключения), вместо этого используются отдельные методы типа init или методы фабрики (фабричного метода).
- Для C++ классов, описывающих интерфейс, должны соблюдаться требования:

    - Не имеет определенных конструкторов.

    - Содержит только публичные «чистые» виртуальные методы =0 (исключая деструктор).

    - Не содержит полей (data members).

    - Содержит виртуальный деструктор.

```
class IDemo
{
    public:
```

```
        virtual ~IDemo() {}
        virtual void method() = 0;
};
```

- Помечать методы классов, которые не меняют состояние объекта как const double getValue() const {return value;}
- Все входные параметры, передаваемые в функции как ссылки, должны передаваться как const[i]:

```
void Foo (const string &in1, const vector<int> &in2);
```

- Необходимо передавать входной параметр как указатель, если необходимо передать nullptr (const *T in);
- Следует избегать преобразований типов C (C cast), используйте вместо преобразования C++ (static_cast, const_cast, reinterpret_cast).
- Не следует использовать dynamic_cast, используйте qobject_cast для QObjects.
- Необходимо использовать конструктор для преобразования типа для простых типов: int(myFloat) вместо (int)myFloat.
- Необходимо использование преинкремента и декремента ++i и --i вместо (i++ и i--).
- Для forward declaration следует использовать строго тот же тип. Не следует использовать forward declaration «class» для «struct» и «struct» для «class». (не объявлять class B; если определено как struct B {…}).
- Для объявления сигнала Qt следует использовать макрос Q_SIGNAL(S).
- Для описания Qt соединения сигнал-слот следует использовать connect с типизированными методами:

```
connect(const QObject *sender, PointerToMemberFunction signal, const QObject *receiver, PointerToMemberFunction method,
Qt::ConnectionType type)
connect(const QObject *sender, PointerToMemberFunction signal, Functor functor)
connect(const QObject *sender, PointerToMemberFunction signal, const QObject *context, Functor functor, Qt::ConnectionType type)
```

вместо:

```
connect(const QObject *sender, const char *signal, const QObject *receiver, const char *method, Qt::ConnectionType type)
connect(const QObject *sender, const char *signal, const char *method, Qt::ConnectionType type)
```

- Не прописывать using namespace <name> в заголовочных файлах в глобальном пространстве имен.

Базы данных

- В качестве первичного ключа используются универсальный уникальный идентификатор UUID, для UUID используется тип и алгоритм GUID (глобальный уникальный идентификатор), не допускается генерация «в ручную» случайными числами, в базах данных, не поддерживающих тип GUID используется строковый тип. Для базы SQLite3 – создается custom тип «GUID».
- CUD (Create, Update, Delete) операции над логически связанными сущностями(объектами) бизнес-модели ("Has-A Relationship" или "Part Of

Relationship", когда один объект является составной частью другого объекта), должны проводиться в рамках одной транзакции бизнес-логики.

- Транзакции бизнес-логики должны удовлетворять ACID (Atomicity — Атомарность, Consistency — Согласованность, Isolation — Изолированность, Durability — Устойчивость).
- В решениях, архитектура которых предусматривает модуль работы с хранилищем (например, базой) данных (или реализует шаблон «Репозиторий»), доступ к данным в обход этого модуля не допускается.
- Наименования полей-идентификаторов записи таблиц - “Id”. Для внешних ключей к имени поля добавить суффикс «_id»;
- Для реляционных структур данных всегда используются внешние ключи. Во время CUD (Create, Update, Delete) операций над логически связанными сущностями(объектами) не допускается отключение ограничений внешних ключей (foreign keys constraints).

Интерфейс пользователя

- Выравнивание контента для контролов и ячеек таблиц для текста по левому краю, для остального (числа, даты) – по правому краю, если иное не указано в требованиях.
- Для реализации вариантов единственного выбора из статичного множества вариантов, если количество вариантов не превышает 3-х, то не допускается использование выпадающих списков (drop down box), вместо этого предпочтительно использовать Radio Buttons.
- Минимальное использование модальных окон и диалогов.
- Стили шрифтов, иконок, контролов, а также размеры кнопок выполняются в общем стиле в пределах приложения.
- Стили объектов интерфейса приложения должны подчиняться единой конфигурации (общие настройки стилей, минимально возможное применение индивидуальных стилей для отдельных объектов).
- При реализации интерфейса предпочтение отдается стандартным графическим компонентам (в том числе принятыми в Компании).

Репозитории кода

- Исходные коды всех решений храниться в репозиториях кода Компании. Не допускается только локальное (на рабочей станции) хранение кода.
- Рекомендуется для проекта (решения) создавать ветки:

  - default – общая рабочая ветка (ведется разработка);

  - stable – ветка релизов (заливается код протестированного решения);

  - *<featureName>* – ветки для разработки отдельной feature (или нескольких features в этом случае *<changeName>*).

- *<hotFix>* - ветки для исправления решений из ветки релизов. (в качестве наименования fix*<ShortNameOfFix>*)

  Разработка в ветке stable запрещена.

- При наименовании веток используются только латинские буквы и цифры, не используются пробелы и небуквенные символы. Рекомендуется короткие имена для веток (не более 10 символов) в регистре camelCase.
- Синхронизация локальных изменений кода с общими репозиториями выполняется как можно чаще:

  - не реже 1 раза в сутки заливать код в ветку фичи.

  - при изменении общей ветки – затянуть изменения общую ветку в ветку фичи, общая ветка + ветка фичи ->> ветка фичи (проверка наличия изменений не реже 1 раза в сутки).

- Рекомендуется для каждой доработки (реализации feature или bugfix), трудозатратами по времени более суток и\или участии нескольких разработчиков, создается отдельная ветка (feature branch). Для HG – named branch.
- В комментариях при синхронизации с репозиторием (commit, check in) указывается в рамках какой задачи выполняются изменения.

<fix|add|modify|refactor|не писать ничего>: *<краткий комментарий>*
<task number “task description|task name”>
<дополнительное описание> -опционально

Пример :

```
fix: ToolDesigner (remove all ;))
task 43567 “ToolDesigner: failure on add new tool”
```

- Не допускается заливка кода решения, которое не прошло фазу тестирования, в ветку, определенную как «стабильная».

Рекомендации оформления (для C++)

- Имена, представляющие типы, описываются в смешанном регистре, начиная с верхнего (UpperCamelCase or PascalCase):

```
Line, SavingsAccount
```

- Имена классов, описывающих интерфейс, начинаются с верхней «I»:

```
ILine, ISavingAccount
```

- Имена сигналов составляются из:

  - before + сущ.имя+глагол – для сигналов до изменения

beforeLogCreate

- сущ.имя+глагол-ing – для сигналов во время изменения

logCreating

- существительного имени+глагол (-ed|3f) - для сигналов после изменения

logCreated

- Имена переменных описываются в смешанном регистре, начиная с нижнего (lowerCamelCase or camelCase).

line, savingsAccount

- Именованные константы (включая значения перечислений) записываются в верхнем регистре с нижним подчеркиванием в качестве разделителя.

MAX_ITERATIONS, COLOR_RED, PI

- Названия методов и функций описываются как глаголы, в смешанном регистре и начинаться с нижнего.

getName(), computeTotalWidth()

- Названия пространств имен записывать в нижнем регистре (для разделителей использовать подчеркивание «_», но использовать имя как можно короче, не составляя из нескольких слов).

model::analyzer, io::iomanager, common::math::geometry, data_handling

- Не используется транслитерация наименований на Русском языке латинскими символами (включая комментарии кода):

~~imyaFayla~~ fileName;

- Члены классов (поля, свойства, методы ..) должны быть сгруппированы по виду («поле», «свойство», «метод» ..) и по видимости (public, private ..).
- Комментарии в коде на Английском языке:

  - Не допускается использовать специальные символы (например, TAB) и разрывы страниц в файлах исходного кода.

  - В файлах кода используется кодировка utf-8, если иное не указано в требованиях или Координатором.

- Не рекомендуется писать код (блоки if или циклов) в одну строку.

- Не рекомендуется отмечать специально приватные поля классов префиксами типа «m_» или постфиксами «_», вместо это используйте this-> или self. для доступа из методов класса к приватным полям. Как исключение, возможно использование префикса «_».

- При объявлении типов - указателей символ «*» ставиться к имени переменной:

```
double *scale;
```

- Скобка { с начала строки.

- Пробелы в выражениях a = b + c; и в аргументах функции a = sum(b, c)

- Большие логические выражения разбиваются на несколько строк. Следует использовать логические переменные для вычисления частей сложного логического выражения.

```
bool isFinished = (elementNo < 0) || (elementNo > maxElement);
bool isRepeatedEntry = elementNo == lastElement;

if (isFinished || isRepeatedEntry)
   {
     :
   }
```

Документирование

- Комментарии в соответствии правилами doxygen.
- Экспортируемые функции, экспортируемые классы, публичные методы экспортируемых классов, интерфейсы, документируются: как минимум описывается назначение и параметры.
- Использование и функционирование сторонних библиотек документируется (в части сторонних компонентов: что нужно установить и как настроить чтобы собрать проект и что и как должно быть установлено и настроено для работы решения).
- Все структуры данных (записи, контракты данных, DTO объекты) и единицы измерения описаны (назначение структуры данных, констант и в каких единицах используемые в них числовые значения).

Правила контроля качества кода (Code review)

- В процессе контроля не проверяются рекомендации (такие как «Рекомендации оформления»).
- Контрольный лист для проверки кода как минимум должен включать пункты:

  - Присутствуют «Магические числа» или строковые константы в коде.

  - Присутствует ли избыточный или повторяющийся код.

- Есть ли закомментированный код.

- Присутствуют ли неиспользуемые части кода, оставшиеся после рефакторинга или изменения функционала («Лодочный якорь»).

- Является ли код независимым, насколько это возможно.

- Все ли входные данные проверяются (на корректный тип, длину, формат, диапазон).

- Обрабатываются ли ошибки при использовании сторонних утилит.

- Обрабатываются ли неверные значения параметров.

- Все ли экспортируемые функции, публичные методы и классы прокомментированы.

- Использование и функционирование сторонних библиотек документируется.

- Все структуры данных и единицы измерения описаны.

1.3. Тестирование

Тестирование продукта осуществляется тестировщиками Отдела программного обеспечения. В «МИКС» используются виды тестирования, согласно описанию ниже:

Порядок тестирования	**Вид тестирования**
Тестирование функционала каждого реализованного изменения	функциональное
Тестирование после устранения дефекта	Функциональное, проверочное
Тестирование «кандидата на реализацию» после реализации всех изменений (функций)	Регрессионное
Тестирование релиза	Дымное (облегченный тест на работоспособность)

Дефектом считается отклонение от заявленных функциональных или нефункциональных возможностей программного обеспечения.

Описание дефекта должно содержать:

- Наименование и версию программного обеспечения;
- Краткое описание дефекта;
- Шаги для воспроизведения дефекта;
- Описание полученного результата;
- Описание ожидаемого результата;
- Дополнительное приложение: данные проекта(ов) MAXIM при работе с которыми был обнаружен дефект.

Программное обеспечение готово к выпуску, если оно имеет статус «кандидат для выпуска», а также не имеет известных критических дефектов, для известных значительных дефектов описаны способы их обхода.

2. Процессы совершенствования и поддержки ПО «DiaLog»

2.1. Менеджмент конфигурации ПО

Модернизация и обновление ПО выполняется Отделом программного обеспечения, в составе:

- Руководитель проектов – 1 чел.,
- системный аналитик – 3 чел.,
- специалист технической поддержки – 1 чел.,
- разработчик – 12 чел.,
- тестировщик – 2 чел.

Руководитель проектов:

Обязанности:

1. Управление зарегистрированными Заказчиком(ами) запросами на изменения, организация работ по реализации изменений (от принятия решения о реализации или отклонении запроса Заказчика на изменение до его реализации и доведении до Заказчика)
2. Оценка затрат на реализацию изменений
3. Постановка задач на реализацию изменений проектной команде

Компетенции:

1. Развитые презентационные навыки
2. Коммуникабельность
3. Знание гибких методологий разработки (Scrum)
4. Знание основных принципов работы компонентов информационных систем (БД, сервер приложения, веб-сервер)
5. Умение описывать и моделировать автоматизируемые бизнес-процессы

Разработчик:

1. Исполнение задач на реализацию изменения(ий) согласно Политикам правилам разработки
2. Написание технической документации
3. Знание среды разработки Visual Studio, Qt Creator
4. Знания C++, SQL, Python
5. Проектирование API

6. Разработка архитектуры, дизайна решения
7. Сборка и развертывание программного решения

Тестировщик:

1. Контроль качества ПО и его компонентов
2. Тестирование API, пользовательского интерфейса,
3. Регрессионное тестирование
4. Регистрация ошибок, в случае если ПО не удовлетворяет качеству и сценариям проверки
5. Написание автотестов
6. Знание теории и методики тестирования ПО
7. Составление плана тестирования и написание тест-кейсов

Аналитик:

1. Сбор требований заказчиков
2. Формирование технического описания задачи на основе информации, полученной от Заказчика
3. Осуществление приемки задачи после реализации запроса на изменение
4. Оптимизация бизнес-процесса Заказчика при анализе запроса на изменение
5. Подготовка пользовательской документации

2.2. Процесс решения проблем в ПО, устранение неисправностей

Неисправности могут быть выявлены тестировщиком Отдела программного обеспечения либо непосредственно Пользователем в ходе эксплуатации продукта.

При выявлении неисправностей тестировщиком последний самостоятельно регистрирует в адрес разработчика задачу с описанием проблемы в ERP системе, определяет и указывает в задаче приоритет по ней. Разработчик берет задачу в работу, согласно приоритетам.

Описание стадий процесса при выявлении неисправностей при работе в ПО представлено ниже:

№	Стадия	Описание
1.	Создание запроса	Пользователи (лог-инженеры и аналитики-интерпретаторы) в процессе работы в DiaLog сталкиваются с ситуациями, затрудняющими рабочий процесс. Исходя из ситуации, пользователями производится формирование запроса. Запросы регистрируются пользователем через любой из доступных каналов связи. Полученный по телефону

		запрос фиксируется в Teams инженером технической поддержки.
2.	Уточнение информации	На этом этапе производится определение приоритета запроса по мнению пользователя и уточнение информации у пользователя инженером технической поддержки.
3.	Обработка критического обращения [Опционально]	При поступлении критического обращения, требующего незамедлительного решения, инженер технической поддержки подключает на звонок с пользователем одного или нескольких экспертов проекта Maxim для решения запроса за максимально короткий срок. Критичность обращения определяется пользователем: 1. Пользователи не могут продолжать рабочий процесс 2. Пользователь не может получить результат самостоятельно за согласованный с заказчиком период времени
4.	Определение типа запроса (Классификация), регистрация запроса	На этом этапе дополнительно подключается тестировщик для поиска сценария возникновения ошибки в программе. Когда запрос понятен и классификация очевидна, инженер технической поддержки производит регистрацию задачи в ERP системе.
5.	Консультация	После регистрации запрос может быть закрыт инженером технической поддержки, если его можно решить консультацией и вопрос пользователя будет удовлетворен.
6.	**6.1** Эскалация запроса (на 2-ю линию поддержки) **6.2** Анализ запроса	Если запрос не может быть закрыт, то для решения подключается вторая линия – для анализа, поиска причины возникновения запроса, и определения сроков реализации.
7.	**7.1** Эскалация запроса (на 3-ю линию поддержки) **7.2** Формирование задач для реализации изменений	Если запрос не может быть закрыт без изменений в коде программы, он эскалируется дальше на 3-ю линию разработчикам. Приоритет по реализации изменений определяется третьей линией вместе с координатором проекта и согласовывается с ключевыми пользователями.

8.	Закрытие запроса (Перевод запроса в статус «Done»)	Реализация изменений, устраняющих ошибку программы, или реализация дополнительных функций программы.
9.	Формирование инструкций и пополнение базы знаний	После реализации изменений запрос может быть возвращен обратно на первую линию для предоставления ответа пользователю в удобном и понятном формате в виде краткой или подробной инструкции.

2.3. Техническая поддержка пользователей

Техническая поддержка имеет трехуровневую структуру – 1-я, 2-я и 3-я линия:

- Специалистом первой линии является инженер технической поддержки. Он производит обработку писем, звонков, сообщений в мессенджерах. Инженер технической поддержки консультирует пользователей, уточняет необходимую информацию для предоставления решения и создает пользовательские инструкции с целью формирования базы знаний.
- Вторая линия состоит из тимлидеров групп тестирования, разработки, и аналитики. Они подключаются, когда сотрудник первой линии не может решить запрос, и занимаются предоставлением решения, если задача не требует проверки и изменения кода программы.
- Третья линия — это разработчики. Сотрудники, которые должны внести ряд изменений в существующий код или написать новый для решения запроса пользователя.

Коммуникация конечных пользователей со специалистами технической поддержки осуществляется посредством электронного почтового сервиса. Для приема обращений - DiaLog.Support@miksmp.ru.

Запросы обрабатываются в рабочее время (с 09:00 до 17:00 по МСК), критические запросы обрабатываются 24/7 сразу в момент поступления.