Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____А.А.Замышляева
«____»_____2018 г.

Рефакторинг программы автосопровождения летательных аппаратов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2018.88.ПЗ ВКР

Руководитель работы, доцент
_____/А.К.Демидов
«____»_____2018 г.
Автор работы
Студент группы ЕТ-414
_____/ Е.Н.Гусев
«____»_____2018 г.
Нормоконтролер, доцент
_____/Т.Ю.Оленчикова
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Гусев Е.Н. Рефакторинг программы автосопровождения летательных аппаратов. – Челябинск: ЮУрГУ, ЕТ-414, 48 с., 6 ил., библиогр. список – 20 наим., 2 прил.

Цель данной работы – разработать новую версию программы автосопровождения летательных аппаратов, путем проведения рефакторинга программного кода. В рамках дипломной работы были рассмотрены существующие проблемы программного кода, рассмотрены основные методы рефакторинга, были сформированы рекомендации к написанию кода, произведен ряд работ для подготовки к проведению рефакторинга, после чего был выполнен рефакторинг программного кода.

Результатом дипломной работы является обновленная версия программы автосопровождения, полностью сохраняющая свой функционал. За исключением исправленных и переработанных алгоритмов.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Ежедневно в мире совершается около 85 000 гражданских авиарейсов. За годы прогресса и инноваций существенно увеличилось количество и качество перелетов. Данная ситуация достигается за счет успешного производства разноплановых локаторов, а также многолетнего труда конструкторов и инженеров.

В реальном времени за всеми перелетами следят диспетчеры, но каким образом? Диспетчер наблюдает за воздушной обстановкой на экране и в случае необходимости управляет и корректирует движение летательных аппаратов. Но откуда появляется информация на экране? Именно этим занимается программа автосопровождения летательных аппаратов. Она решает проблему автоматизации обмена информацией, а также формирования треков полета самолета, корректировку полетной информации, фильтрации отраженных сигналов и многое другое.

АО НИИИТ–РК более 25 лет занимается задачами обеспечения диспетчеризации полетов. За эти годы были достигнуты большие результаты в области обеспечения автоматизации процесса сопровождения летательных аппаратов. Но на данный момент качество и способы написания программного кода значительно продвинулись вперед. Возникает необходимость переработки кода программного обеспечения с целью обеспечения стабилизации сопровождения и упрощения дальнейших модернизаций.

Рефакторинг это процесс изменения внутренней структуры программного обеспечения. Сферой применения рефакторинга является исходный код, основной целью которого является упрощение будущих модификаций и облегчение понимания кода человеком. Индикатором необходимости применения рефакторинга является «проблемный код», но, в каждом конкретном случае для вынесения окончательного решения о необходимости рефакторинга необходимо произвести анализ.

Рефакторинг необходимо применять тогда, когда добавляется новая функция, исправляется ошибка или проводится разбор кода. Для корректного применения рефакторинга, необходимо знание существующих методов рефакторинга.

В процессе применения методов рефакторинга необходимо производить качественное тестирование каждого отдельно взятого шага. Но даже при применении модульных тестов нельзя утверждать об отсутствие ошибок в программе, возникших в процессе рефакторинга.

Достоинствами проведения рефакторинга является следующее:

- улучшение композиции программного обеспечения;
- облегчение понимания программного обеспечения;
- помощь в нахождении ошибок;
- ускорение написания программ;
- сохранение работоспособности программы.

Недостатками проведения рефакторинга является следующее:

- требование о наличии предварительных тестов;
- проблемы отсутствия доказательной верификации корректности изменений.

1 АНАЛИЗ ТРЕБОВАНИЙ. АНАЛИЗ СУЩЕСТВУЮЩИХ ПРОБЛЕМ ПРОЕКТА И МЕТОДЫ ИХ РЕШЕНИЯ.

1.1 Постановка задачи

Поставлена задача разработать новую версию программы автосопровождения летательных аппаратов, путем проведения рефакторинга программного кода. В процессе модернизации необходимо сохранить внешнее поведение программного обеспечения, но улучшить внутреннюю структуру. Программа автосопровождения летательных аппаратов предназначена для организации выполнения приема входной информации, обработку и формирование треков сопровождения, с последующей выдачей потребителям в различных форматах.

Цели проведения рефакторинга:

- улучшить композицию программного обеспечения;
- облегчить понимание программного обеспечения;
- ускорить процесс модернизации программного обеспечения;
- организовать внутреннее тестирование программного обеспечения;
- устранить существующие ошибки программного обеспечения.

Для достижения поставленной цели необходимо решить следующие задачи:

- разработать требования к приложению;
- изучить и проанализировать современные технологии для проведения рефакторинга;
- разработать модульные тесты;
- произвести рефакторинг к существующим алгоритмам программы;
- произвести тестирование программы.

1.2 Описание предметной области

Программа автосопровождения выполняет прием отметок, сопровождение треков и выдачу потребителям.

Операционная система QNX – это UNIX–подобная операционная система реального времени, основанная на микроядре и передаче сообщений [1].

В процессе работы программы автосопровождения используются механизм разделяемой памяти и очередей сообщений.

Механизм разделяемой памяти реализован в администраторе процессов и предназначен для быстрого обмена большими объемами данных между процессами. Суть этого механизма заключается в том, что процесс запрашивает администратор процессов о выделении некоторого региона памяти. Затем этот регион может отображаться разными процессами на их собственное адресное пространство [2].

Очереди сообщений POSIX реализованы в QNX с помощью администратора очередей mqueue. Очереди сообщений – это именованные объекты, поэтому данный механизм можно использовать для обмена между процессорами одной ЭВМ, так и между процессами, работающими на разных узлах сети [3].

Выдача информации потребителям происходит по протоколу Астерикс. Астрекс – это протокол прикладного уровня, ответственный за определение и сбор данных, разработанный для обеспечения трансляции и обмена данными наблюдения.

Модульное тестирование – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея 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 [4].

1.3 Анализ требований к программе

1.3.1 Общие требования к программе

Поставлена задача произвести рефакторинг программного кода программы автосопровождения. Целями проведения рефакторинг ПО являются:

- улучшить композицию программного обеспечения;
- облегчить понимание программного обеспечения;
- ускорить процесс модернизации программного обеспечения;
- организовать внутреннее тестирование программного обеспечения;
- устранить существующие ошибки программного обеспечения.

1.3.1.1 Требования к функциональным характеристикам

Программное обеспечение должно обладать следующими функциями:

- работать под управлением ОС QNX 6.5;
- в процессе работы программы необходимо производить чтение, обработку и выдачу данных в режиме реального времени;
- программа автосопровождения должна производить непрерывную работу в фоновом режиме;
- программа автосопровождения должна взаимодействовать с сервисной программой отладки;
- программа автосопровождения должна производить непрерывное резервирование;
- программа автосопровождения должна производить запись отладочной информации;

– программа автосопровождения должна производить фильтрацию, сглаживание, экстраполяцию, а также определять участки отраженных сигналов;

– программа автосопровождения должна осуществлять обмен данными по существующим протоколам.

1.3.1.2 Требования к режиму функционирования ПО

В программа автосопровождения должны присутствовать следующие режимы функционирования:

- режим отладки;
- рабочий режим.

Режим отладки должен быть предназначен для отладки алгоритмов и должен выполняться на этапе ввода в эксплуатацию. В этом режиме должно происходить журналирование необходимым компонентов программы, отлаживание реакций на происходящие воздействия.

Рабочий режим должен быть единственным допустимым режимом штатного функционирования системы. В рабочий режим система должна переводиться после режима отладки, когда становится ясно, что система функционирует верно, корректно отрабатываются необходимые шаблоны.

1.3.2 Требования к надежности

1.3.2.1 Требования к диагностированию ПО

Реализовать возможность ведения журнала действий пользователей (логов) с целью выявления несанкционированных вызовов, ошибок входной и выходной информации, различных аварийных ситуаций. В этом случае в журнал должны заноситься данные и сведения о действиях программы.

1.3.2.2 Требования к информационной безопасности

Защита информации от несанкционированного доступа обязана удовлетворять следующим требованиям:

- доступ к программе автосопровождения необходимо осуществлять только в случае, если введен пароль;
- программные средства защиты не должны реализовываться в ущерб основным функциональным характеристикам ПО, таким как быстродействие и надежность.

1.3.2.3 Обработка отказов программы и аварийных ситуаций

Отказы программы вследствие некорректных действий пользователя при взаимодействии с приложением недопустимы. Отказ программы автосопровождения может быть вызван из-за выхода из строя вычислителя навигации. В этих случаях необходимо, чтобы все функции основного

вычислителя взял на себя резервный. Резервный вычислитель должен продолжить корректную работу.

1.3.4 Перспективы модернизации и развития

При разработке приложения необходимо учесть возможность дальнейшего совершенствования ПО, а именно добавление новых участков алгоритмов, обновление существующих.

1.4 Анализ существующих проблем проекта

На протяжении долгих лет над программой автосопровождения трудились множество специалистов. Каждый из которых вносил свой вклад в развитие программы автосопровождения, дорабатывая или обновляя существующие алгоритмы. В конечном итоге получилась работоспособная программа, но в процессе эксплуатации возникают ситуации, в которых программа автосопровождения нестабильна. В таких случаях производятся доработки по исправлению ошибок, из–за которых это могло возникнуть.

1.4.1 Осложнения, вызванные выбором языка программирования

В данной реализации программы процесс доработки сильно затруднен, ввиду использования разных стилей программирования. Изначально программа разрабатывалась в процедурном стиле на языке программирования C, но некоторые доработки велись уже с использованием объектно–ориентированного подхода на языке программирования C++. Таким образом, получился проект, в котором присутствуют как функции длинною до десяти тысяч строк кода, с загруженной логикой, так и объекты, выполняющие свой ограниченный функционал.

1.4.2 Неиспользуемый код

В процесс модернизации программного кода, программисты комментируют старые участки кода, а на их месте пишут новую программную реализацию. Также встречаются случаи, когда старый алгоритм остается в проекте, но после его выполнения, происходит выполнение нового алгоритма, тем самым выполняется ряд действий в пустую. Вследствие чего, функции, использующиеся в старой реализации, остаются в проекте. Отсюда появляется проблема, больших и запутанных участков кода, которые никаким образом не влияют работу программы в целом.

1.4.3 Дублирование кода

Во многих алгоритмах существуют вычисления, которые повторяются в нескольких местах. Таким образом, в процессе доработки или корректировки этого участка, приходится находить все вхождения этого участка кода в

программу в целом. И отсюда вытекают ошибки в реализации, которые сложно найти, что в целом пагубно влияет на время модернизации программы. Объединение этих участков кода позволит улучшить программный код и упростить процесс модернизации.

1.4.4 Расходящиеся модификации

В процессе модернизации программы невозможно определить участок, который необходимо обновить. Проблемы появляется в следствие плохой структурированности программы.

1.4.5 Длинные методы

В данной реализации в проекте существуют подпрограммы размерами до десяти тысяч строк, что сильно затрудняет процесс чтения и понимания работы программы. Легко заметить, что чем меньше размер метода, тем легче воспринимается информации заключённая в нем.

1.4.6 Длинный список параметров

В данной реализации проекта, существуют проблемы с передачей параметров в подпрограмму. Логические трудно понимаемый список параметров, различные критерии, признаки. Вся необходимая информация для работоспособности программ автосопровождения, в целом должна храниться в разделяемой памяти. Таким образом, количество дополнительных параметров минимизируется. Большое количество параметров затрудняет процесс использования их в подпрограммах, а также осложняет процесс вызова.

1.4.7 Реализация процесса компиляции и сборки

В проекте единицы трансляции обмениваются информацией путем связывания по имени, используя ключевое слово `extern`. Для упрощения согласования разных единиц трансляции необходимо использовать механизм заголовочных файлов, заключающийся в объявлении четкого интерфейса. Также пагубное влияние на процесс компиляции оказывает большое количество неиспользуемых включений.

В проекте существует большое количество констант описанных при помощи директив `#define`, которые замедляют процесс компиляции и загружают пространство имен.

Макросы и символические константы, унаследованные из языка C, при написании программ на C++ необходимо избегать. Вместо символических констант предпочтительнее использовать `const` [5].

1.4.8 Отсутствие модульных тестов

В проекте отсутствуют модульные тесты, и нет возможности написания. В целом используется около десяти основных функций, размер которых не менее тысячи программных строк. Таким образом, производить тестирование настолько сильно загруженных функций проблематично. В процессе рефакторинга будет производиться тестирование двух версий программ, путем запуска на задокументированной радиолокационной информации. Также в процессе рефакторинга будут создаваться модульные тесты для новой версии программы. Тем самым блоки кода будут протестированы на основе предполагаемых значений методом «белого ящика», а также на основе метода «черного ящика» с использования задокументированной радиолокационной информации.

Тестирование чёрного ящика – метод тестирования функционального поведения объекта с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве тестируемого объекта. Стратегия поведенческого теста исходит из технических требований и их спецификаций [6].

Тестирование белого ящика – структурное тестирование, которое учитывает внутренние механизмы системы или компонента [4].

1.4.9 Заключение

С каждым годом происходит непрерывный процесс модернизации программы автосопровождения, а в связи с этим возрастает количество проблем в проекте. Перечислив все недостатки проекта можно сделать вывод о необходимости проведения рефакторинга, с целью улучшения качества программного кода.

1.5 Анализ существующих методов решения проблем проекта

Рефакторинг – это процесс изменения внутренней структуры программного обеспечения, имеющее целью облегчить понимание его работы и упростить модификацию, не затрагивая наблюдаемого поведения. Процесс рефакторинга происходит путем минимального сопротивления. Необходимо производить модификации к небольшим частям, с целью облегчения нахождения ошибок. Таким образом, при последовательном рефакторинге произойдет существенное улучшение программного 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.

В [7] предлагается использовать описанные далее методы.

1.5.1 Встраивание временной переменной

Данный метод используется тогда, когда у нас существует временная переменная, которой один раз присваивается простое выражение. Локальные переменные затрудняют выделение, тем самым мешают проведению рефакторинга. Необходимо произвести замену всех ссылок на данную переменную этим выражением. Таким образом, избавляемся от промежуточного кода, а также появляется возможность использования данного метода в других методах.

Например, такой участок кода

```
int isLinearMovement(     MapMemAcS::_tr*   track,
                          unionMark*        mark )
{
   int linearMovement = 0;
   double diffRange = abs( track->range - mark->range );
   if( diffRange <= getStrobeForLinearMovement( track ) ) {
         linearMovement = 1;
}
   return linearMovement;
}
```

преобразуется к виду

```
int isLinearMovement(     MapMemAcS::_tr*   track,
                          unionMark*        mark  )
{
   return      abs( track->range - mark->range )   <=
               getStrobeForLinearMovement( track );
}
```

1.5.2 Введение поясняющей переменной

Данный метод используется в случае, если существует некоторое сложное выражение, и при анализе метода является трудным для чтения. Необходимо поместить части выражения во временную переменную, имя которой поясняет его назначение. В такой ситуации полезно с помощью временных переменных превратить выражение в нечто, лучше поддающееся управлению. Упрощается чтение и понимание кода. Примером служит длинный алгоритм, в котором каждый шаг можно раскрыть с помощью временной переменной.

Например, такой участок кода

```
if (      ( reflectorAreas[ i ].isUse == 1 )              &&
          ( azimuth >= reflectorAreas[ i ].azimuthStart   &&
          azimuth <= reflectorAreas[ i ].azimuthEnd )     &&
          ( rang >= reflectorAreas[ i ].rangeStart        &&
          range <= reflectorAreas[ i ].rangeEnd    )      )
{
   return 1;
}
```

преобразуется к виду

```
bool     isUseReflectorAreas     = ( reflectorAreas[ i ].isUse == 1 );
bool     isInAzimuthStrobe =     azimuth >= reflectorAreas[ i ].azimuthStart &&
                                 azimuth <= reflectorAreas[ i ].azimuthEnd;
bool     isInRangeStrobe   =     range >= reflectorAreas[ i ].rangeStart  &&
                                 range <= reflectorAreas[ i ].rangeEnd;
if (     isUseReflectorAreas     &&
         isInAzimuthStrobe       &&
         isInRangeStrobe   )
{
   return 1;
}
```

1.5.3 Замена переменной вызовом метода

Проблема с переменными в том, что они временные и локальные. Поскольку они видны лишь в контексте метода, в котором используются, временные переменные ведут к увеличению размеров методов, потому что только так можно до них добраться. После замены временной переменной методом запроса получить содержащиеся в ней данные может любой метод класса. Это существенно содействует получению качественного кода для класса.

Простыми случаями данного рефакторинга являются такие, в которых присваивание временным переменным осуществляется однократно, и те, в которых выражение, участвующее в присваивании, свободно от побочных эффектов. Остальные ситуации сложнее, но разрешимы. Облегчить положение

могут предварительное использование метода «Расщепление временной переменной».

Например, такой участок кода

```
void selectCoefficientForFiltering ( MapMemAcS::_tr* track )
{
   track->alfa = ( 2.0 * ( 2.0 * n + 1.0 ) ) / ( ( n + 2.0 ) * ( n + 1.0 ) );
   track->beta = 6.0 / ( ( n + 2.0 ) * ( n + 1.0 ) );
   ...
}
```

преобразуется к виду

```
void selectCoefficientForFiltering ( MapMemAcS::_tr* track )
{
   track->alfa = getAlfa();
   track->beta = getBeta();
   ...
}
private double getAlfa(){
   return      ( 2.0 * ( 2.0 * _timeOverview + 1.0 ) ) /
               ( ( _timeOverview + 2.0 ) * ( _timeOverview + 1.0 ) );
}
private double getBeta(){
   return ( 6.0 / (  ( _timeOverview + 2.0 ) *
                     ( _timeOverview + 1.0 ) ) );
}
```

1.5.4 Расщепление поясняющей переменной

Данный метод используется в случае, если имеется временная переменная, которой неоднократно присваивается значение, но это не переменная цикла и не временная переменная для накопления результата. Если использовать переменные для различных несвязанных целей, существует большая вероятность появления ошибок в тот момент, когда будут производится какие–то изменения в коде, связанные с этими переменными. Придётся перепроверять все случаи использования переменной, чтобы удостовериться в отсутствии ошибки в коде. Необходимо создать для каждого присваивания отдельную временную переменную. Использование одной и той же переменной для решения разных задач очень затрудняет чтение кода.

Например, такой участок кода

```
double  delta = track->azimuth – mark-> azimuth;
isInStrobeAzimuth ( delta );
delta   =     track-> range – mark-> range;
isInStrobeRange ( delta );
```

преобразуется к виду

```
double  deltaAzimuth     = track-> azimuth – mark-> azimuth;
isInStrobeAzimuth   ( deltaAzimuth );
double  deltaRange       = track-> range – mark-> range;
isInStrobeRange     ( deltaRange );
```

1.5.5 Удаление присваиваний параметрам

Данный метод используется в случае, если параметру метода присваивается какое–то значение. Множественные присваивания разных значений параметру приводят к тому, что сложно понять, какие именно данные должны находиться в параметре в определенный момент времени. Проблема усугубляется, если параметр и то, что он должен хранить, описаны в документации, но фактически, его значение может не совпадать с ожидаемым внутри метода.

Если параметр передаётся по ссылке, то после изменения его значения внутри метода, оно передается аргументу, который подавался на вызов этого метода. Очень часто это происходит случайно и приводит к печальным последствиям. Код оказывается значительно понятнее, если параметр используется только для представления того, что передано, поскольку это строгое употребление.

Например, такой участок кода

```
void formationTargetDesignation ( double course )
{
   if ( course >     PI ) {
         course = course – PI;
   }
   formationCourse ( course );
   …
}
```

преобразуется к виду

```
void formationTargetDesignation ( double course )
{
   double courseForAircraft = course;
   if ( course >     PI ) {
         courseForAircraft = course – PI;
   }
   formationCourse ( courseForAircraft );
   …
}
```

1.5.6 Выделение метода

Самая важная причина создания метода — снижение сложности программы. Происходит минимизация объема кода, облегчение сопровождения программы и снижение числа ошибок, но без абстрагирующей силы методов сложные программы было бы невозможно охватить умом [8].

Выделение метода заключается в выделении из длинного и/или требующего комментариев кода отдельных фрагментов и преобразовании их в отдельные методы, с подстановкой подходящих вызовов в местах использования. В этом случае действует правило: если фрагмент кода требует комментария о том, что он делает, то он должен быть выделен в отдельный метод. Также правило: один метод не должен занимать более чем один экран (25–50 строк, в зависимости от условий редактирования), в противном случае некоторые его фрагменты имеют самостоятельную ценность и подлежат выделению. Из анализа связей

выделяемого фрагмента с окружающим контекстом делается вывод о перечне параметров нового метода и его локальных переменных.

Локальные переменные действуют только в исходном методе, поэтому при «Выделении метода» с ними связана дополнительная работа. В некоторых случаях они даже вообще не позволят выполнить рефакторинг. Проще всего, когда локальные переменные читаются, но не изменяются. В этом случае можно просто передавать их в качестве параметров. Сложности возникают, когда локальным переменным присваиваются новые значения. Увидев присваивание параметру, нужно сразу применить метод «Удаление присваиваний параметрам».

Часто временных переменных так много, что выделять методы становится очень трудно. В таких случаях необходимо сократить число временных переменных с помощью метода «Замены временной переменной вызовом метода»

Например, такой участок кода

```
int isMovementError(      MapMemAcS::_tr*   track,
                          unionMark*        mark )
{
   …
   meanValue = meanMovement( track );
   di = sqrt(  pow( mark->range, 2.0 ) –
               pow( track->height, 2.0 ) );
   markx = di * cos( mark->azimuth );
   marky = di * sin( mark->azimuth );
   distance = hypot( fabs( track->xs - markx ),
                     fabs( track->ys - marky ) );
   moveError = 0;
   if(   distance > 1.25 * meanValue ||
         distance < 0.75 * meanValue) {
         moveError = 1;
   }
   return moveError;
}
```

преобразуется к виду

```
double distanceBetweenTrackAndMark(   MapMemAcS::_tr*   track,
                                      unionMark*        mark )
{
   double di, markx, marky;
   di = sqrt(  pow( mark->range, 2.0 ) –
               pow( track->height, 2.0 ) );
   markx = di * cos( mark->azimuth );
   marky = di * sin( mark->azimuth );
   return hypot(     fabs( track->xs - markx ),
                     fabs( track->ys - marky ) );
}
int isMovementError(      MapMemAcS::_tr*   track,
                          unionMark*        mark )
{
   meanValue = meanMovement( track );
   distance = distanceBetweenTrackAndMark( track, mark );
   moveError = 0;
   if(   distance> 1.25 * meanValue ||
         distance< 0.75 * meanValue) {
         moveError = 1;
   }
   return moveError;
}
```

1.5.7 Выделение класса

Данный метод используется в случае, если существует некоторый класс выполняющий работу, которую следует поделить между двумя классами. Класс должен представлять собой ясно очерченную абстракцию, выполнять несколько отчетливых обязанность. Необходимо создать новый класс и переместить соответствующие поля и методы из старого класса в новый.

Конкретный класс нацелен на то, чтобы как можно точнее соответствовать частной концепции и стратегии ее реализации. Быть понятным и пригодным к использованию независимо от других классов [9].

Схема выделения класса приведена на рисунке 2.1.

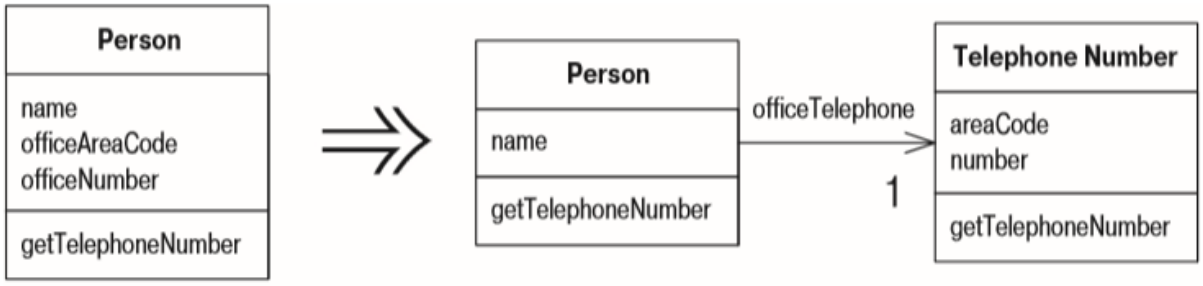

Рисунок 2.1 – Выделение класса

1.5.8 Декомпозиция условного оператора

Данный метод используется в случае, если имеется сложная условная цепочка проверок (if–then–else). Как и в любом большом блоке кода, можно сделать свои намерения более ясными, если выполнить его декомпозицию и заменить фрагменты кода вызовами методов, имена которых раскрывают назначение соответствующего участка кода.

Например, такой участок кода

```
if(      ( track->k3 == 0 )      &&
         ( track->k4 == 0 )      &&
         ( track->isSps == 0 )   ){
   p_kat_048_3_ struct_output;
   struct_output.p_isd = 1;
   struct_output.fx3 = 0;
   ...
} else {
   p_kat_048_4_ struct_output;
   struct_output.fx3 = 1;
   struct_output.p_kr_3a = 1;
   ...
}
```

преобразуется к виду

```
bool isShortMessage () {
   return      ( track->k3 == 0  )      &&
               ( track->k4 == 0  )      &&
               ( track->isSps == 0 );
}
```

```
void creature_shortProfileCategory() {
   p_kat_048_3_ struct_output;
   struct_output.p_isd = 1;
   struct_output.fx3 = 0;
   ...
}
void creature_longProfileCategory () {
   p_kat_048_4_ struct_output;
   struct_output.fx3 = 1;
   struct_output.p_kr_3a = 1;
   ...
}
if ( isSmallMessage() ){
   creature_shortProfileCategory ();
} else {
   creature_longProfileCategory();
}
```

1.5.9 Консолидация условного выражения

Данный метод используется в случае, если существует ряд проверок условий, дающих одинаковый результат. Необходимо объединить их в одно условное выражение и выделить его.

Например, такой участок кода

```
int OrganizationMessageTRACK () {
   if(   ( track_number <= 0 ) ||
         ( track_number >= SMES ) ) {
         return 0;
   }
   if( track->pz == 0 ) {
         return 0;
   }
   if(   ( track->smoothedRange  >= maxRange ) ||
         ( track->range          >= maxRange ) ) {
         return 0;
   }
   …
}
```

преобразуется к виду

```
bool isExitCodingTRACK () {
   if( ( track_number <= 0 ) || ( track_number >= SMES ) ) {
         return true;
   }
   if( track->pz == 0 ) {
         return true;
   }
   if(   ( track->smoothedRange  >= maxRange ) ||
         ( track->range          >= maxRange ) ) {
         return true;
   }
   return false;
}
int OrganizationMessageTRACK () {
   if ( isExitCodingTRACK() ) return 0;
   …
}
```

1.5.10 Замещение алгоритма

Для каждой конкретной задачи существует множество методов решения. Они обладают своими преимуществами и недостатками, которые определяют, насколько тот или иной метод подходит для решения поставленной задачи. Решение, которое казалось вполне приемлемым на бумаге, на практике может оказаться ошибкой [10].

Если обнаруживается более понятный способ сделать что либо, следует заменить сложный способ простым. Рефакторинг позволяет преобразовывать сложные вещи на более простые части. Иногда наступает такой момент, когда надо взять алгоритм целиком и заменить его чем–либо более простым. Например, когда происходит подробный анализ задачи и обнаруживается, что существует более простой способ организации выполнения алгоритма. Распространённым случаем использования замещения алгоритма, является замена алгоритма на функцию из библиотек.

Иногда, когда желательно изменить алгоритм, чтобы он решал несколько иную задачу, легче сначала заменить его таким кодом, в котором потом проще произвести необходимые изменения. Перед выполнением этого приема необходимо убедиться, что дальнейшая декомпозиция метода уже невозможна. Замену большого и сложного алгоритма выполнить очень трудно. Только после его упрощения замена становится осуществимой.

Например, такой участок кода

```
struct Sector {
   int   count;
   track [ MAX_COUNT_TRACK];
   }
Sector sector;
...
for (int i = 0; i < MAX_TRC; i++) {
   if( MapAcS->tr[i].pz != 0 && MapAcS->tr[i].pzr == 0 ) {
         if(   ( MapAcS->tr[i].smoothedAzimuth >= AzimuthLeft ) &&
               ( MapAcS->tr[i].smoothedAzimuth < AzimuthRight ) ) {
               sector.count++;
               sector.track[ sector.count ].Azimuth =
                                          MapAcS->tr[i].smoothedAzimuth;
               sector.track[ sector.count ].Range =
                                          MapAcS->tr[i].smoothedRange;
               sector.track[ sector.count ].numKs = i;
         }
   }
}
bool f = true;
while ( f ) {
   f = true;
   for ( int i = 0; i < sector.count - 1; i++ ){
         if (  sector.track[ i ].Azimuth > sector.track[ i + 1 ].Azimuth ) {
               track temp;
               temp = sector.track[ i + 1 ];
               sector.track[ i + 1 ] = sector.track[ i ];
               sector.track[ i ] = temp;
               f = false;
         }
   }
```

```
}
for ( int i = 0; i < sector.count ; i++ ){
   asterix.codingTRACK(    sector.track[ i ].numKs,
                           &MapAcS->tr[ sector.track[ i ].numKs ] );
}
```

преобразуется к виду

```
std::list <AvInSector> AvInSecList;
for( int i = 0; i < MAX_TRC; i++ ) {
   if( MapAcS->tr[i].pz != 0 && MapAcS->tr[i].pzr == 0 ) {
         if(   ( MapAcS->tr[i].smoothedAzimuth >= AzimuthLeft ) &&
               ( MapAcS->tr[i].smoothedAzimuth < AzimuthRight ) ) {
               AvInSector AvInSect;
               AvInSect.Azimuth = MapAcS->tr[i].smoothedAzimuth;
               AvInSect.Range = MapAcS->tr[i].smoothedRange;
               AvInSect.numKs = i;
               AvInSecList.push_back( AvInSect );
         }
   }
}
AvInSecList.sort( sortAzimuth );
for (    std::list< AvInSector >::iterator iter = AvInSecList.begin();
         iter!= AvInSecList.end();
         ++iter      )
{
   asterix.codingTRACK(    iter->numKs,
                           &MapAcS->tr[ iter->numKs ] );
}
```

1.5.11 Замена магического числа символической константой

Магические числа — это числовые значения, встречающиеся в коде, но при этом неочевидно, что они означают. Затрудняется понимание программы и усложняется процесс рефакторинга.

Дополнительные сложности возникают, когда нужно поменять определённое магическое число. Это нельзя сделать автозаменой, т.к. одно и то же число может использоваться для разных целей, а значит, необходимо будет проверять каждый участок кода, где используется это число. Необходимо создать константу, дать ей имя, соответствующие смыслу, и замените ею число**.**

Например, такой участок кода

```
creature_typeMessage( 0x01 );
azimuth = track->markAzimuth    * 360.0 / 65536;
```

преобразуется к виду

```
static const int    TYPE_SEVER  = 0x01;
static const double CMR_AZIMUTH = 360.0 / 65536;
creature_typeMessage( TYPE_SEVER );
azimuth = track->markAzimuth    * CMR_AZIMUTH;
```

1.5.12 Инкапсуляция поля

Одной из концепций объектного программирования является инкапсуляция или возможность сокрытия данных объекта. Иначе все данные объектов были бы публичными, и другие объекты могли бы получать и модифицировать данные

вашего объекта без его ведома. При этом отделяются данные от поведений, связанных с этими данными, ухудшается модульность частей программы и усложняется её поддержка.

Например, такой участок кода

```
class StageDecoding {
   …
   private int _rcvShortMsgCount,
               _rcvLongMsgCount;
}
void decoding_message( Msg *ptr ) {
   if ( _rcvShortMsgCount > _rcvLongMsgCount )
   …
}
```

преобразуется к виду

```
class StageDecoding {
   …
   private int _rcvShortMsgCount,
               _rcvLongMsgCount;
}
int getCountShortMsg()     { return _rcvShortMsgCount; };
int getCountLongMsg()      { return _rcvLongMsgCount; };
void decoding_message( Msg *ptr ) {
   if ( getCountShortMsg() > getCountLongMsg() )
   …
}
```

1.5.13 Преобразование процедурного проекта в объекты

Главным недостатком процедурного подхода заключается в том, что процессы и данные существуют отдельно друг от друга. В объектно-ориентированном подходе основная категория объектной модели – класс. Он объединяет в себе на элементарном уровне как данные, так и операции, которые над ними выполняются (методы). Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения. Классы позволяют проводить конструирование из полезных компонентов, обладающих простыми инструментами, что дает возможность абстрагироваться от деталей реализации. Инкапсуляция информации защищает наиболее критичные данные от несанкционированного доступа. Именно с этой точки зрения изменения, связанные с переходом от процедурного к объектно-ориентированному подходу, являются наиболее заметными.

Объектно-ориентированные системы более открыты и легче поддаются внесению изменений, поскольку их конструкция базируется на устойчивых формах. Это дает возможность системе развиваться постепенно и не приводит к полной ее переработке даже в случае существенных изменений исходных требований.

Данный метод используется в случае, если существует код, написанный в процедурном стиле. Данный метод относится к разряду крупных рефакторингов, а именно трудоёмкость метода гораздо выше, методов, описанных ранее, это следует учитывать при выборе.

Для преобразования процедурного проекта в объекты, необходимо преобразование записей данных в объекты, разделение поведения, перемещение поведения в объекты. К каждой длинной процедуре необходимо применять сопутствующие рефакторинги, чтобы разбить ее на части. Схема преобразования процедурного проекта в объекты приведена на рисунке 2.2.

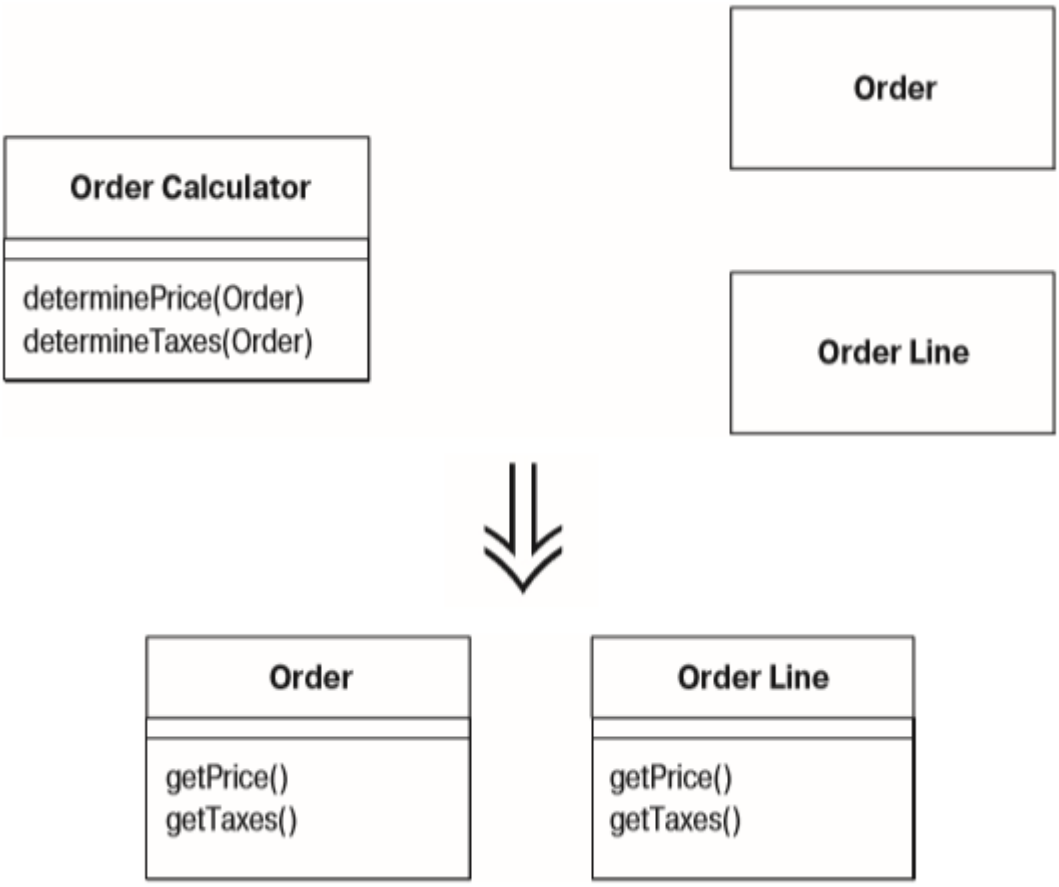

Рисунок 2.2 – Преобразование процедурного проекта в объекты

1.5.14 Заключение

В данном подразделе были приведены основные методы рефакторинга. Подробно описаны случаи необходимости их применения, а также непосредственно примеры их применения на основе программы автосопровождения летательных аппаратов.

1.6 Рекомендации к написанию кода

Проекты существуют долгие годы, и над его работоспособностью трудятся большое количество разработчиков. Каждый программист индивидуален и придерживается тех правил, которые он построил для себя. В данном проекте все преобразования производились при помощи конкретных правил, далее подробное приведем каждое из них. Это поможет в дальнейших преобразованиях или доработках сохранить однотипный стиль программирования в рамках данного проекта.

1.6.1 Общие соглашения об именовании

– Необходимо использовать английский язык, при выборе имени идентификатору.

– Именовать типы данных необходимо в смешанном регистре, начиная с верхнего.

```
Filtration, ReflectorArea
```

– Именовать переменные необходимо в смешанном регистре, начиная с нижнего

```
filtration, reflectorArea
```

– Именовать константы необходимо в верхнем регистре с нижним подчёркиванием в качестве разделителя.

```
CMR_TIME, BYTE_SYNX, TYPE_KAT_253
```

– Методы и функции необходимо именовать глаголами, записанными в смешанном регистре, начиная с нижнего.

```
getLenghtMessage(), getStrobeForLinearMovementManeuver()
```

– Переменным–итераторам следует давать имена i, j, k и т. д.

– Методы возвращающие логический тип данных необходимо именовать с префикса is.

```
isFirstPrimary(), isGpsTimeSourceCorrect()
```

– Необходимо избегать сокращения в именах.

1.6.2 Файлы исходных кодов

– Необходимо использовать защиту от повторного включения заголовочных файлов.

```
#ifndef FILTRATION_H_
#define FILTRATION_H_
…
#endif
```

– Необходимо сортировать и группировать директивы включения, располагать в начале файла.

```
#include <algorithm>
#include <iostream.h>

#include "Asterix.h"
#include "RliReserve.h"
#include "SlidingBuffer.h"
#include "StageTracking.h"

#include "share/AvInSector.h"

#include "Connect/ConnectDebugLog.h"
#include "Connect/Connect.h"
```

1.6.3 Выражения

– Инициализацию переменных необходимо производить в месте их объявления.

```
double deltaAzimuth;
deltaAzimuth = fabs( track->azimuth - mark->azimuth );
```

– Необходимо избегать использования глобальных переменных.

– Каждая переменная должна нести определенный смысл.

– Области видимости каждой переменной должен быть минимален.

– Символ указателя или ссылки необходимо ставить после имени типа.

```
unionMark* mark
```

– Переменные, относящиеся к циклу, необходимо инициализировать непосредственно перед ним.

– В конструкции цикла необходимо указывать выражение, непосредственно относящиеся к управлению циклом.

– Необходимо бесконечный цикл записывать в виде формы `while(true)`.

– Необходимо избегать сложных условных выражений.

– Необходимо избегать исполнимых выражений в условных операторах.

– Необходимо избегать использования неопределенных чисел в коде. Необходимо объявлять их, как именованные константы.

– Необходимо использовать выравнивание везде, где возможно улучшить читаемость.

```
void ProccessingHeight(   MapMemAcS::_tr*   track,
                          unionMark*        mark  );
PxConfigReadDoubleCx( ctx, "sett", "DKM_M", 810.0,      &configuration.DKM_M );
PxConfigReadDoubleCx( ctx, "sett", "DDKM_M",1000.0,     &configuration.DDKM_M);
PxConfigReadDoubleCx( ctx, "sett", "AKM_M", 4.4,        &configuration.AKM_M );
```

– Необходимо явно указывать модификаторы доступа `public`, `protected` и `private`.

1.6.4 Заключение

Рекомендации к написанию кода помогут в дальнейшем существовании проекта производить обновление или корректировку, поддерживая единообразную стилистику. При этом упрощается понимание существующего кода проекта.

2 ПРИМЕНЕНИЕ МЕТОДОВ РЕФАКТОРИНГА НА ПРОЕКТЕ

Перед проведением работ по рефакторингу проекта, необходимо произвести максимальное возможное разделение на этапы, тем самым, производя модернизацию последовательно. Отлаживая и проверяя каждое совершенное действие.

В данной реализации проекта, рефакторинг будет производиться по этапам работы самой программы автосопровождения. То есть, узнавая из документации алгоритм и описанные действия в нем, необходимо выявить по всему проекту подпрограммы, участвующие в соответствующих блоках алгоритма. Выявить и устранить участки мертвого кода, упрощая дальнейший процесс рефакторинга.

Мертвый код, называемый также недостижимым кодом, представляет собой блок кода, который по тем или иным причинам, в зависимости от логики алгоритма, действительно недостижим и бесполезен [11].

Схема алгоритма обработки РЛИ приведена на рисунке 4.1.

2.1 Тестирование

Тестирование программного обеспечения — это процесс анализа или эксплуатации программного обеспечения с целью выявления дефектов [12].

При применении методов рефакторинга важным предварительным условием является наличие надежных тестов. Необходимо создавать автоматические тесты, которые проверяют собственные результаты. Тестирование не выявит все ошибки, но поможет избежать большинства.

Для каждого дополнения или модификации системы необходимо понимать все хитросплетения кода. Каждое изменение, вносимое в код, нарушает работу кода в двух–трех местах. Ни одно изменение не проходит тривиально [13].

Таким образом, количество и качество написанных автоматических тестов сильно влияет на процесс модернизации программного продукта. Графическое представление цикла проведения рефакторинга приведено на рисунке 4.2.

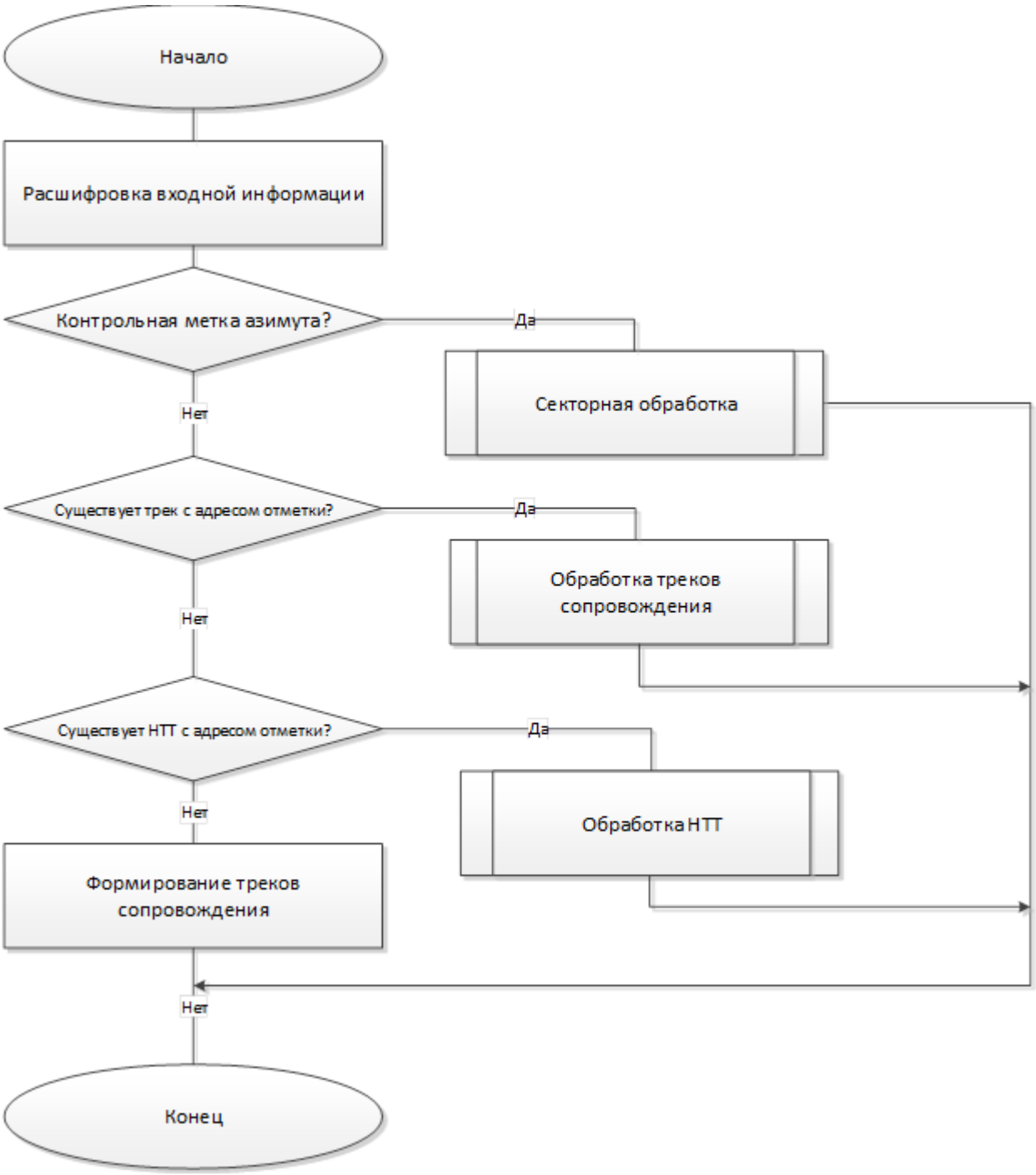

Рисунок 4.1 – Схема алгоритма обработки РЛИ

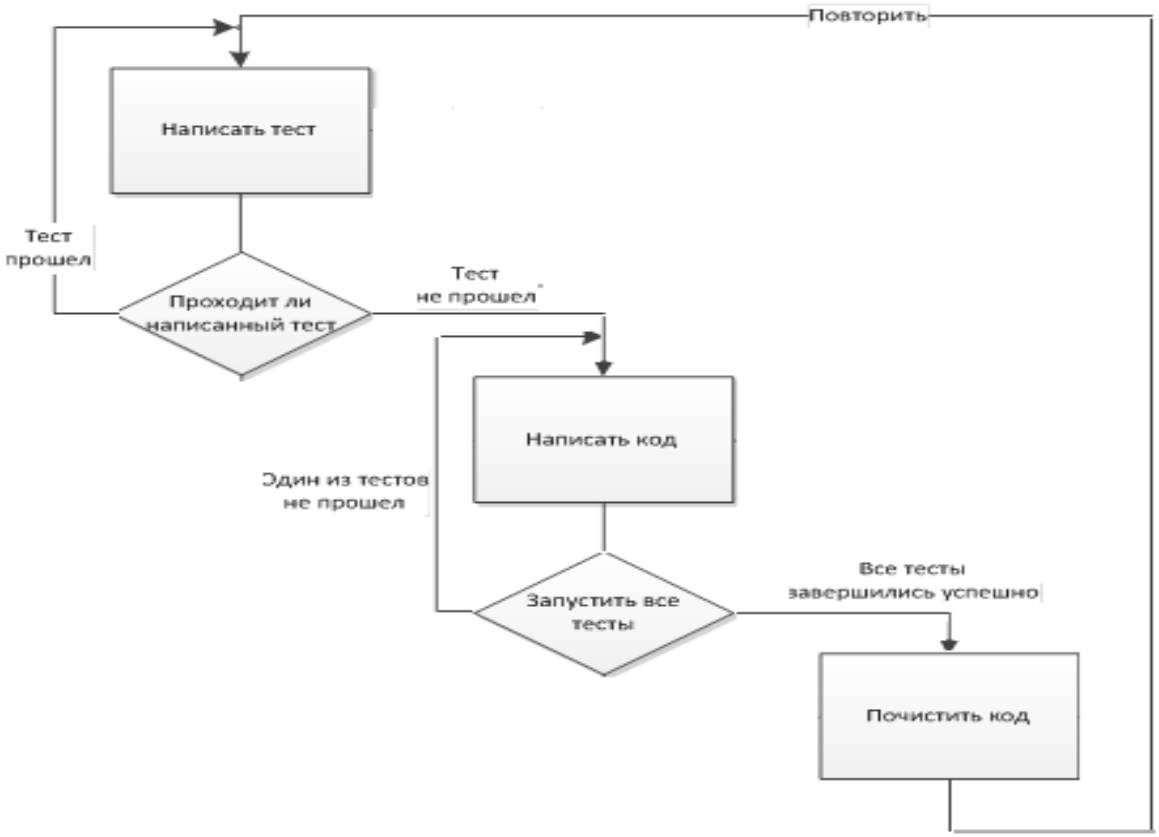

Рисунок 4.1 – Схема алгоритма цикла проведения рефакторинга

В данной реализации программы автосопровождения отсутствовали автоматизированные тесты. Google C++ Testing Framework – библиотека для модульного тестирования на языке C++. Google Test построена на методологии тестирования xUnit, то есть когда модули программы (классы, функции) проверяются отдельно друг от друга.

Таким образом, перед применением методов рефакторинга были написаны модульные тесты. В виду плохой структурированности программы, не для всех алгоритмов программы они были созданы. Также по ходу проведения рефакторинга необходимо поддерживать их работоспособность. Приведем пример модульных тестов до начала и после завершения рефакторинга для алгоритма фильтрации координатной информации содержащейся в треке сопровождения.

До

```
TEST( abFilterTesting , smoothedCoord_0_5 )
{
   Map_Of_Mem_Ac::_tr  track;
   track.markX = track.markY = 2500;
   track.x = track.y = 1000;
   track.alfa = 0.5;
   Filtration( &track , 100 );
   EXPECT_EQ(1750, track.xs );
   EXPECT_EQ(1750, track.ys );
}
```

После

```
TEST( abFilterTesting , smoothedCoord_0_5 )
{
   ReflectorArea reflectorArea;
   Filtration filtration;
   filtration.initialization( &reflectorArea );
   MapMemAcS::_tr    track;
   track.markX = track.markY = 2500;
   track.x = track.y = 1000;
   track.alfa = 0.5;
   filtration.starting( &track , 10 );
   EXPECT_EQ(1750, track.xs );
   EXPECT_EQ(1750, track.ys );
}
```

Пример тестирования алгоритма фильтрации координатной информации

```
[==========] Running 23 tests from 4 test cases.
[----------] Global test environment set-up.
[----------] 6 tests from FilterTesting
[ RUN      ] FilterTesting.smoothedCoord_0_5
[       OK ] FilterTesting.smoothedCoord_0_5 (0 ms)
[ RUN      ] FilterTesting.smoothedCoord_0_2
[       OK ] FilterTesting.smoothedCoord_0_2 (0 ms)
[ RUN      ] FilterTesting.smoothedCoord_0_8
[       OK ] FilterTesting.smoothedCoord_0_8 (0 ms)
[ RUN      ] FilterTesting.smoothedAzimuth
[       OK ] FilterTesting.smoothedAzimuth (10 ms)
[ RUN      ] FilterTesting.smoothedRange
[       OK ] FilterTesting.smoothedRange (0 ms)
[ RUN      ] FilterTesting.smoothedVelocity
[       OK ] FilterTesting.smoothedVelocity (0 ms)
[----------] 6 tests from FilterTesting (10 ms total)

[----------] 6 tests from selectCoefficientForFilteringTesting
[ RUN      ] selectCoefficientForFilteringTesting.zeroCount
[       OK ] selectCoefficientForFilteringTesting.zeroCount (0 ms)
[ RUN      ] selectCoefficientForFilteringTesting.threeCount
[       OK ] selectCoefficientForFilteringTesting.threeCount (0 ms)
[ RUN      ] selectCoefficientForFilteringTesting.mtNoManeuver
[       OK ] selectCoefficientForFilteringTesting.mtNoManeuver (0 ms)
[ RUN      ] selectCoefficientForFilteringTesting.mtPotentialManeuver
[       OK ] selectCoefficientForFilteringTesting.mtPotentialManeuver (0 ms)
[ RUN      ] selectCoefficientForFilteringTesting.mtManeuver
[       OK ] selectCoefficientForFilteringTesting.mtManeuver (0 ms)
[ RUN      ] selectCoefficientForFilteringTesting.mtHiManeuver
[       OK ] selectCoefficientForFilteringTesting.mtHiManeuver (0 ms)
[----------] 6 tests from selectCoefficientForFilteringTesting (0 ms total)

[----------] 6 tests from checkEndManeuverTesting
[ RUN      ] checkEndManeuverTesting.zeroCount
[       OK ] checkEndManeuverTesting.zeroCount (0 ms)
[ RUN      ] checkEndManeuverTesting.notStrobAzimut
[       OK ] checkEndManeuverTesting.notStrobAzimut (0 ms)
[ RUN      ] checkEndManeuverTesting.notStrobDalnost
[       OK ] checkEndManeuverTesting.notStrobDalnost (0 ms)
[ RUN      ] checkEndManeuverTesting.mtHiManeuver
[       OK ] checkEndManeuverTesting.mtHiManeuver (0 ms)
[ RUN      ] checkEndManeuverTesting.mtManeuver
[       OK ] checkEndManeuverTesting.mtManeuver (0 ms)
[ RUN      ] checkEndManeuverTesting.mtPotentialManeuver
```

```
[       OK ] checkEndManeuverTesting.mtPotentialManeuver (0 ms)
[----------] 6 tests from checkEndManeuverTesting (1 ms total)

[----------] 5 tests from checkStartManeuverTesting
[ RUN      ] checkStartManeuverTesting.mtPotentialManeuver
[       OK ] checkStartManeuverTesting.mtPotentialManeuver (0 ms)
[ RUN      ] checkStartManeuverTesting.mtManeuver
[       OK ] checkStartManeuverTesting.mtManeuver (0 ms)
[ RUN      ] checkStartManeuverTesting.mtNoManeuver
[       OK ] checkStartManeuverTesting.mtNoManeuver (0 ms)
[ RUN      ] checkStartManeuverTesting.isStrobe_mtNoManeuver
[       OK ] checkStartManeuverTesting.isStrobe_mtNoManeuver (0 ms)
[ RUN      ] checkStartManeuverTesting.isMove_mtNoManeuver
[       OK ] checkStartManeuverTesting.isMove_mtNoManeuver (0 ms)
[----------] 5 tests from checkStartManeuverTesting (0 ms total)

[----------] Global test environment tear-down
[==========] 23 tests from 4 test cases ran. (13 ms total)
[  PASSED  ] 23 tests.
```

2.2 Подключение к разделяемой памяти

В программе автосопровождения, все информация о летательных аппаратах, режимах работы изделия, номера работающего комплекта, и других необходимых параметров хранится в разделяемой памяти. Тем самым, для использования и корректировки данных необходимо обеспечить соединение.

В предыдущей версии программы, этот процесс производился путем подключения к разделяемой памяти, и последующей пересылкой указателя на нее, во все необходимы подпрограммы. В данной реализации, программы было принято решение, реализовать одиночный класс, отвечающий за все подключения к разделяемой памяти.

Если предполагается только один объект класс, необходимо использовать образец проектирования одноэлементных множеств [14].

Тем самым класс одиночка, гарантирует наличие единственного экземпляра класса и предоставляет глобальную точку доступа к нему, обеспечивается надежность подключения.

Были выделены публичные поля класса, для хранения подключения к разделяемой памяти.

```
PTRMAINSTRUCT  MapMain;
PTRMAPMEMACS   MapAcS;
PTRREGSTRUCT   MapReg;
PTRKARTASTRUCT MapKarta;
PTRPRO1STRUCT  MapPro;
PTRDNVRLSTRUCT MapDnvrl;
```

Также были выделены приватные методы класса, для организации подключения.

```
void connectToMapMain();
void connectToMapAcS();
void connectToMapReg();
void connectToMapPro();
void connectToMapKarta();
void connectToMapDnvrl();
```

Был создан статичный метод, предоставляющий глобальную точку доступа к этому экземпляру.

```
static connectSharedMemory& GetConnect(){
        static connectSharedMemory object;
        return object;
}
```

Следующим преобразованиям при использовании разделяемой памяти, является преобразование обработки данных связанных с треком или отметкой. Заменить передачу индекса на трек или отметку на указатель. Тем самым при проведении подобного рода рефакторинга, упрощается наглядность кода и безопасность данных (исключается возникновение ошибок выхода за пределы массива).

Например, такой участок кода

```
int isLinearMovementForManeuver( int track, int mark )
{
   float diffRange = ( float )abs(     Map_Ac->tr[ track ].d –
                                       Map_Ac->nak[ mark ].d );
   …
}
```

преобразуется к виду

```
int isLinearMovementForManeuver(      MapMemAcS::_tr*   track,
                                      unionMark*        mark )
{
   double diffRange = abs( track->range - mark->range );
   …
}
```

2.3 Подключение к дескрипторам очередей

Программа автосопровождения работает в режиме реального времени, таким образом необходимо поддерживать стабильный обмен сообщениями между другими программами комплекса. В целях обеспечения надежности процесса обмена сообщениями, входящие сообщения прежде записываются в очередь, а затем происходит обработка на стадии расшифровки.

Было принято решение произвести выделение класса, отвечающего за хранение дескрипторов очередей и организацию подключения/переподключения.

Были выделены приватные поля класса, хранящие дескрипторы очередей и методы для их получения.

```
mqd_t    query_ac_s
mqd_t    query_net;
void getDescriptorQuery_AC_S();
void getDescriptorQuery_Net();
```

Также было произведено выделение методов организации подключение и переподключения.

```
void connectToQuery_AC_S();
void connectToQuery_Net();
void reconnectToQuery_AC_S();
void reconnectToQuery_Net();
```

2.4 Чтение конфигурационного файла

В момент загрузки главная управляющая программа считывает файлы с конфигурационными параметрами. Все параметры, которые могут задаваться в этом файле, имеют и внутренние предопределенные значения в самой программе. То есть с помощью файла конфигурации можно изменять параметры настройки, находящиеся в самой программе.

В процессе работы программы автосопровождения необходимо производить считывание конфигурационных параметров. Для этих целей было произведено выделение класса `ConfigurationFileReader`.

Далее был выделен метод организующий чтение конфигурационного файла.

```
readConfiguration();
```

Также в процессе чтения необходимо проверять корректность конфигурационных параметров, для этих целей был выделен метод.

```
void checkValue( ConfigurationParameters &configuration );
```

2.5 Отладочная информация

В процессе работы программы автосопровождения происходит множество корректировок полетной информации, в целях подробного отслеживания процессов обработки необходимо производить документирование.

Было произведено выделение класса отладочной информации. Данный класс был реализован как одиночный. В процессе работы которого создается один выходной поток и происходит последовательная запись в него.

Был выделен приватный метод класса, для подключения к отладочному файлу.

```
void connectToFile();
```

Также были выделены публичные поля и методы класса, для хранения потока и организации записи в него.

```
ofstream *outBuffer;
void printf_log( ofstream* out, const char* s, ... );
```

В дальнейшем происходили преобразования, связанные с редактированием выходного потока данных. А именно, для наглядности логов, было принято решение сначала указывать какая именно подпрограмма производит запись строки, а далее уже следует непосредственно отладочная информация. Таким образом улучшается возможность нахождения необходимого участка отладочной информации.

Пример отладочного файла

```
12:18:56:188837 main ---> QueueThread -> Прием сообщения ret = 40
12:18:56:188837 StageDecoding -> decoding_message : Количество сообщений = 1
12:18:56:189837 StageDecoding ---> decoding_message : mark type = 5
12:18:56:189837 StageDecoding -> decoding_message_KMA : 13
12:18:56:189837 StageIdentification -> Отметки проходят ассоциацию с треками
12:18:56:189837 StageIdentification ---> Не найден трек с адресом 3038
12:18:56:189837 StageIdentification -> Отметки проходят ассоциацию с НТТ
12:18:56:191837 StageIdentification ---> Запись отметки в массив НТТ
12:18:56:191837 StageIdentification ---> ProcessingTrackWithoutKO
12:18:56:192837 StageIdentification ---> AcceleratedDispatch  numSektor 11
12:18:56:192837 Asterix -> codingKMA
12:18:56:192837 Asterix ---> creature_cadreHome         size = 5 lenght 5
12:18:56:192837 Asterix ---> creature_dataSource        size = 1 lenght = 6
12:18:56:192837 Asterix ---> creature_sourceIdentifier  size = 2 lenght = 8
12:18:56:192837 Asterix ---> creature_typeMessage       size = 1 lenght = 9
12:18:56:193837 Asterix ---> creature_numberSector      size = 1 lenght = 10
12:18:56:193837 Asterix ---> creature_stateOfTheSystem  size = 3 lenght = 13
12:18:56:193837 Asterix ---> creature_cadreEnd          size = 4 lenght = 17
12:18:56:193837 Asterix ---> synchronizationRepeating   size = 0 lenght = 17
12:18:56:193837           10 02 22 00 0b d4 50 00 02 60 88 02 20 47 0f 10
12:18:56:194836           03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
12:18:56:194836 Asterix -> write_messageAsterix
12:18:56:201835 Asterix ---> write_messageAsterix -> vo_vrl   : msg ret = 41
12:18:56:201835 Asterix ---> write_messageAsterix -> query_net: msg ret = 41
12:18:56:421802 ServiceInfo ---> Update -> TimerTics = 14
```

2.6 Сервисное окно

В процессе работы программы автосопровождения происходит множество корректировок полетной информации, в целях отслеживания процессов обработки в режиме реального времени необходимо производить обновление информации в сервисном окне программы автосопровождения.

Для этих целей было произведено выделение класса, который содержит публичные методы инициализации подключения и организации обновления сервисного окна.

```
void Initialization( StageDecoding* _stageDecoding );
void Update();
```

Пример сервисного окна программы автосопровождения приведен на рисунке 4.3.

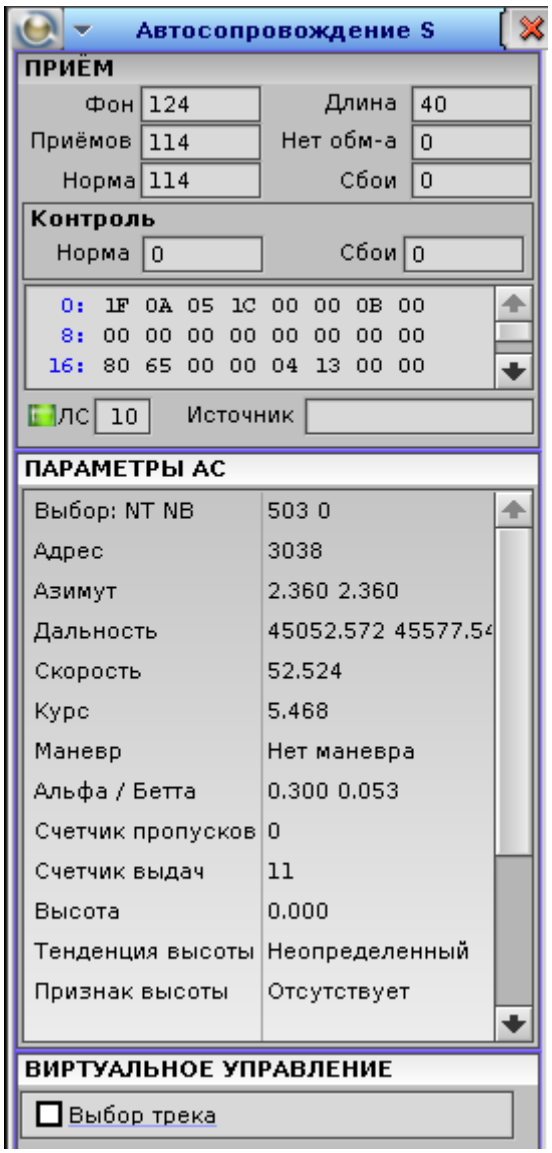

Рисунок 4.3 – Сервисное окно

2.7 Этап расшифровки и предварительная обработка принятной порции сообщений

На вход в программу автосопровождения поступают сообщения, подразделенные на подтипы. В соответствии с этим было произведено выделение методов. Название методов отражают расшифровку определенного подтип входящего сообщения, тем самым упрощается нахождение необходимого участка кода.

```
void decoding_message_11();
void decoding_message_4_5();
void decoding_message_20_21();
void decoding_message_KMA();
```

Ранее обработка производилась таким путем

```
switch ( Kac_Qms.msg_tip & 0x3f ) {
   case 0x7:
         if( ( Kac_Qms.Data[0] & 0x0f ) == 0x04 ) {
         …
         } else if( ( ( Kac_Qms.Data[0] & 0x0f ) == 0x01 ) ||
         ( ( ( Kac_Qms.Data[0] & 0x0f ) == 0x02 ) {
         …
   }
   …
   case 0xA:
   …
}
```

После преобразований, данный код выглядит таким образом

```
switch ( ptr11->type ) {
   case 0x7 :
         decoding_message_11();
   break;
   case 0xA :
         decoding_message_4_5();
         break;
   case 0xB :
         decoding_message_20_21();
         break;
   case 0x5 :
         decoding_message_KMA();
         break;
   default  :
         break;
}
```

Для отображения информации в сервисном окне и отладки путем записи в отладочный файл, были выделены приватные поля класса, отвечающие за подсчет количества определенного вида сообщений. Также созданы методы для получения их количества. Тем самым произвели инкапсуляцию полей.

```
int      rcvMsgCount,
         rcvOwerallMsgCount;
int getCountMsg()          { return rcvMsgCount; };
int getCountOwerallMsg()   { return rcvOwerallMsgCount; };
```

В процессе обработки информации необходимо производить преобразования координатной информации, а именно принятый код необходимо преобразовать в вещественное число. Такие вычисления повторяются во всех выделенных методах расшифровок, таким образом было принято решение, преобразовать структуру входящих сообщений с помощью метода выделения подкласса, в данном случае, подструктуры.

```
struct CommonPart {
   …
   unsigned short    azimuth;
   unsigned short    range;
   double getAzimuth() {
         return azimuth * CMR_AZ_PROS_S * M_PI / 180.0;
   }
   double getRange() {
         return range * CMR_RANGE_PROS_S;
   }
```

```
}
struct OverallAnsw : CommonPart {
   …
}
struct ShortAddrAnsw : CommonPart {
   …
}
```

В сообщениях с подтипом равным 4 и 20 содержится информация о высоте летательного аппарата, но в зависимости от типа высоты, производится различная расшифровка, тем самым было произведено выделение метода.

```
void decoding_message_4_5() {
   decodeHeight(df5->ID, mark);
   …
}
void decoding_message_20_21() {
   decodeHeight(df21->ID, mark);
   …
}
void decodeHeight(unsigned short code, unionMark* mark)
{
   // кодирование высоты в метрах
   if (maskAC->M == 1) {
         codeh = (   ( code & 0x1F80 >> 1) |
                     ( code & 0x3F ) );
         height = codeh * 8.0;
         mark->height = height;
         mark->typeHeight = 1;
   } else if ( maskAC->d1 == 1) {
   // кодирование высоты с ЦМР 25 футов
         codeh =     ( code & 0x1F80 >> 2) |
                     ( code & 0x0020 >> 1 ) |
                     ( code & 0x000F );
         height = (codeh * CMR_HEIGHT_25F ) / CMR_METR_FUT;
         mark->height = height;
         mark->typeHeight = 2;
   } else {
   // кодирование высоты 100 футов, аналогично режиму С
         codeh = (   ( code & 0x1F80 >> 1) |
                     ( code & 0x3F ) );
         height = (  CMR_HEIGHT_25F * heightGilheimGray( codeh )) /
                     CMR_METR_FUT;
         mark->height  = height;
         mark->typeHeight = 3;
   }
}
```

2.8 Этап отождествления координатных отметок

Данный этап осуществляется после предварительной обработки отметки. Тем самым на этап отождествления поступает накопитель отметок, который необходимо обработать. Произвести ассоциацию с треками сопровождения, после с массивом не отождествившихся отметок, а после оставшиеся отметки добавить в массив не отождествивших отметок. Также произвести обработку ранее не обработанных треков сопровождения и произвести ускоренную выдачу этого сектора. Обработка радиолокационной информации производится с отставанием на два сектора.

Данные участок алгоритма ранее находился на этапе расшифровки, тем самым нагружая его логику. Его было решено вынести в отдельный класс, посредством метода «Выделение класса». Подготовить к этому действию функцию необходимо используя методы «Встраивание временной переменной» и «Расщепление поясняющей переменной». Тем самым максимально локализируются производимые операции на этом этапе. Оставшиеся зависимости необходимо будет передавать в качестве параметров.

Следующим действием выделяем подметоды из последовательного алгоритма:

Алгоритм обработки треков в секторе при пропуске отметки

```
void    ProcessingTrackWithoutKO( int numSektor );
```

Алгоритм ускоренной выдачи сектора потребителям

```
void    AcceleratedDispatch( int numSektor );
```

Алгоритм формирования(завязки) трека

```
int     CreateTrack(unionMark* mark, unionNtt* ntt);
```

Тем самым максимально упрощая логику основного управляющего алгоритма.

Запуск этапа отождествления координатных отметок в секторе

```
void    startingProcess ( int numberKMA );
```

2.9 Этап сопровождения треков

На этом этапе происходит обновление полетной информации в треках сопровождения летательных аппаратов. Ранее обработка производилась одной нагруженной функцией. Были принято решение произвести выделение класса этапа сопровождения треков, с последующим выделением методов.

Было выделено два управляющих метода, отвечающих за обработку принятой информации, или, в случае отсутствия данных, отвечающих за обработку пропуска входной информации сектора.

```
void StartingProcess (    MapMemAcS::_tr*   track,
                          unionMark*        mark );
void StartingProcess (    MapMemAcS::_tr*   track );
```

В свою очередь для каждого из них было совершено «Выделение методов». Таким образом, логически разделили последовательной алгоритм, на подметоды.

Сглаживание азимута и дальности при отождествлении трека с отметкой.

```
void SmoothedAzimuthAndRange(   MapMemAcS::_tr*   track,
                                unionMark*        mark );
```

Экстраполяция трека.

```
void ExtrapolationCoordinates(  MapMemAcS::_tr*   track );
```

Процесс обработки высоты трека при отождествлении с отметкой.

```
void ProccessingHeight(   MapMemAcS::_tr*   track,
                          unionMark*        mark );
```

Процесс обработки высоты трека при пропуске отметки.

```
void ProccessingHeight(   MapMemAcS::_tr*   track );
```

Процесс обработки отождествления отметки с треком.

```
void MarkToTrack(    MapMemAcS::_tr*   track,
                     unionMark*        mark );
```

2.9.1 Фильтрация полетной информации

На этапе сопровождения трека был выделен метод, производящий сглаживание координат трека. В виду загруженности метода было принято решение выделить класс, отвечающий за фильтрацию полетной информации.

Под фильтрацией полетной информации подразумевается определение маневра, перерасчет коэффициентов сглаживания, а также непосредственно корректировка полетной информации. Далее было произведено выделение четырех методов.

Метод фильтрации полетной информации.

```
void alphaBetaFilter (     MapMemAcS::_tr*   track ,
                           double            timeOverview );
```

Метод выбора коэффициентов для фильтрации.

```
void selectCoefficientForFiltering( MapMemAcS::_tr* track );
```

Метод определения окончания маневра.

```
int  checkEndManeuver (    MapMemAcS::_tr*   track,
                           unionMark*        mark );
```

Метод определения начала маневра.

```
int  checkStartManeuver ( MapMemAcS::_tr*   track,
                           unionMark*        mark,
                           int               isManeuverMove);
```

2.9.2 Поиск участков отражения сигналов от местных предметов

В процесс обработки информации необходимо учитывать отраженные сигналы от местных предметов. Было принято решение выделить класс поиска участков отражения сигналов.

Информация об участках отраженных сигналах указывается в конфигурационном файле изделия. Тем самым было произведено выделение метода, отвечающего за организацию считывания информации об участках отражения.

```
readReflectorAreas( const char *fileName );
```

Далее выделен метод, отвечающий за проверку трека сопровождения на попадание в участок отраженных сигналов.

```
isTrackInReflectorArea( MapMemAcS::_tr *track );
```

2.9.3 Резервирование полетной информации

В комплексе существует два вычислителя навигации, в которых параллельно работает программа автосопровождения. Один основной, второй резервный. При выходе из строя основного комплекта, происходит переключение на резервный.

Тем самым необходимо осуществлять резервирование полетной информации. При этом главным критерием является то, что номера треков сопровождения должны сохраниться. Для этих целей был выделен класс резервирования полетной информации.

В процессе работы происходит формирование сообщения для резервного вычислителя навигации и выдача в сеть. Было произведено выделение двух методов.

```
void organizationMessage  ( void );
void sendMessage          ( void );
```

Для расшифровки и обработки принятого сообщения был выделен метод.

```
void decodingMessage  ( Msg *ptr );
```

Для возможности отслеживания корректности работы алгоритма, был выделен метод, организующий печать входных/выходных сообщений в отладочный файл.

```
void printfDebugInfo  ( void );
```

2.10 Этап формирования и выдачи выходного сообщения

В процессе работы программы автосопровождения происходит выдача информации потребителям. Для этих целей было произведено выделение класса `Asterix`.

Выдача происходит по протоколу Астерикс (многоцелевой структурированный обмен информацией наблюдения Евроконтроля), тем самым необходимо выделить методы для всех элементов данных каждой категории. Данные методы следует сделать приватными, ведь вызов этих методов происходит из управляющего метода на уровень выше.

Выходная информация программы автосопровождения разделяется на подтипы. В зависимости от сообщения, работают разные участки алгоритмов. Таким образом было произведено выделение четырех публичных методов.

Метод формирования кадра и выдачи по протоколу Астерикс сообщения с треками сопровождения.

```
void codingTRACK ( int _tracknumber, MapMemAcS::_tr* _track );
```

Метод формирования кадра и выдачи по протоколу Астерикс сообщения с кодом отметки азимута (КМА).

```
void codingKMA ( int numberKMA);
```

Метод формирования кадра и выдачи по протоколу Астерикс сообщения север.

```
void codingNorth ( void );
```

Метод формирования кадра и выдачи по протоколу Астерикс сообщения категории 253.

```
void codingMessage253 ( void );
```

При выделении методов возникли проблемы, чтобы выделить методы, необходимо преобразовать глобальные связи, то есть локализовать все

необходимые параметры выдачи. Все данные для выдачи на текущем обзоре должны формироваться на этапе сопровождения трека. Тем самым, необходимо произвести ряд работ, для выявления и устранения глобальных зависимостей.

Также на этом этапе формируется единственное сообщение, информация о котором хранится в приватных полях класса и модифицируется каждым элементом данных. Всего было выделено 27 методов, отвечающих за формирования элементов данных каждой из категорий. Приведем в пример один из них, остальные же методы являются подобными ему.

Метод формирования начального кадра сообщения.

```
void creature_cadreHome (  unsigned char category ) {
   kadr_nome_ struct_output;
   // Формирование кадра
   struct_output.byte_synx = constAsterix::BYTE_SYNX; // Байт синхронизации
   struct_output.byte_home = constAsterix::BYTE_HOME; // Байт-признак начала
   struct_output.kat = category;             // Категория (34 / 48 / 253)
   struct_output.size_box[0] = 0;            // Длина блока данных в байтах
   struct_output.size_box[1] = 0;
   //Запись
   memcpy(     &buffer_outputMessage[ length_outputMessage ],
               &struct_output, sizeof( struct_output ) );
   length_outputMessage += sizeof( struct_output );
   log_printff( DebugLog,
         "Asterix ---> creature_cadreHome size = %d lenght = %d \n",
         sizeof( kadr_nome_ ),
         length_outputMessage );
}
```

Все существующие структуры выходящих сообщений описаны во внешнем файле заголовочном файле. После формирования, выходное сообщение увеличивается строго на размер этого элемента данных. Было выявлено разветвление хранения информации о длине выходного сообщения. Таким образом процесс нахождения размера выходного сообщения сильно загружен. Был применен метод замещения алгоритма.

До

```
lin_k    = lin_k+sizeof( conf_ctrl_s );
lin_bd   = lin_bd+sizeof( conf_ctrl_s );
lin_k2   = isk_10( lin_k, &buff[0], &buff2[0] );
```

После

```
length_outputMessage += sizeof( struct_output );
```

При проведении рефакторинга было выявлено ошибочное формирование некоторых полей элементов данных.

Ошибка

```
if( vib_kom == 0 ) {
   conf_ctrl_s.rdpc=0;    // Выбранный комплект для работы 0-выбран первый
} else {
   conf_ctrl_s.rdpc=1;    // Выбранный комплект для работы 1-выбран второй
}
```

Решение

```
conf_ctrl_s.rdpc = MapMain->workComplex;
```

В данном примере видно, что номер выбранного комплекта напрямую зависит от значения, записанного в переменную vib_kom. Но логическая ошибка заключается в том, что данный критерий на протяжении всей работы программы не изменяется. Из чего следует, что при переключении работы на резервный комплект, выходные данные будут не соответствовать реальным данным.

Констнатные значения использующиеся на этапе формирования и выдачи выходного сообщения были выделены в отдельное пространство имен, описанные в заголовочном файле класса.

```
namespace constAsterix {
   static const double     CMR_TIME    = 1.0 / 128.0;
   static const double     CMR_HEIGHT  = 7.62;
   …
}
```

2.11 Организация модулей программы

В процессе проведения рефакторинга, также были преобразованы модули программы автосопровождения.

Ранее проект состоял из файлов

```
Kac1.cpp
Kac2.cpp
```

Теперь имя файла отображет содержание модуля

```
Filtration.cpp
ReflectorArea.cpp
```

2.12 Выводы по разделу

В процессе проведения рефакторинга, было выделено множество классов. Каждый класс отвечает за определенный этап процесса автосопровождения летательных аппаратов, и выполняет строго отведенный для функционал. Таким образом, проект стал объектно-ориентированным.

Диаграмма разработанных классов приведена на рисунке 4.4.

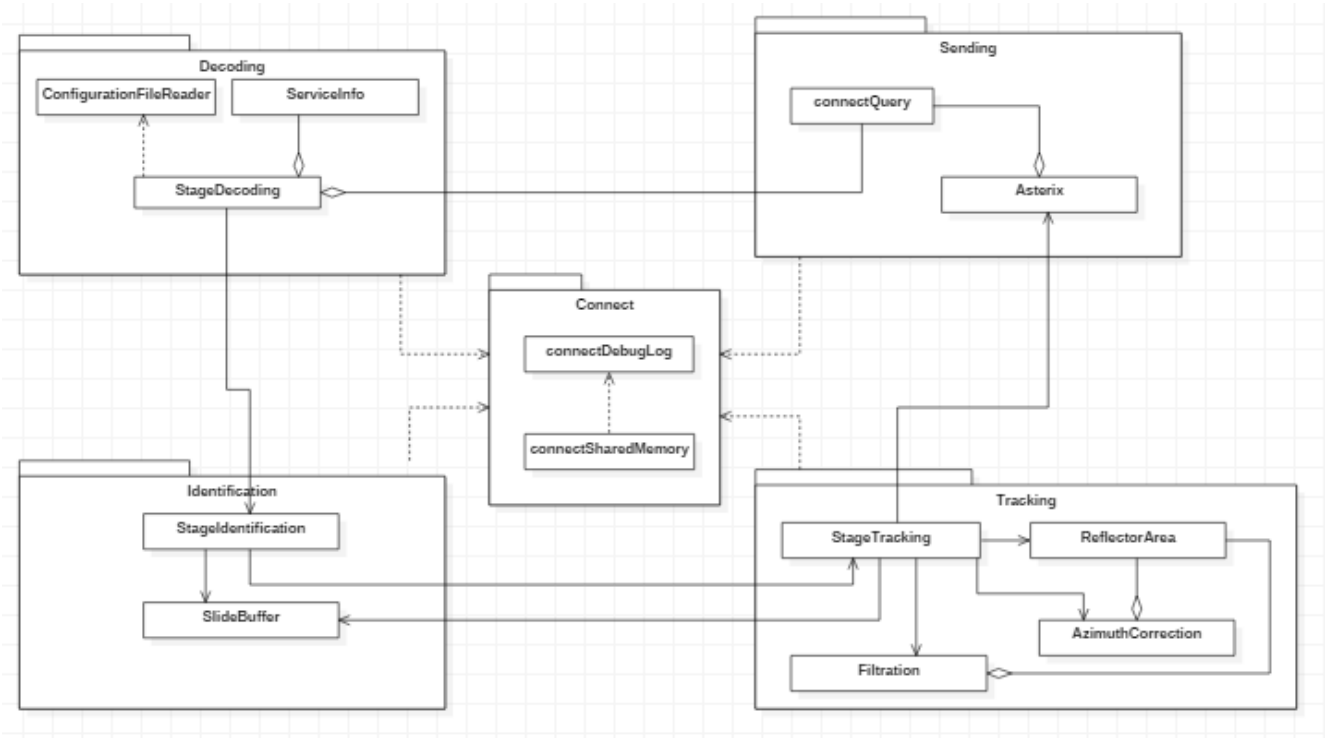

Рисунок 4.4 – Диаграмма классов

ЗАКЛЮЧЕНИЕ

В данной работе были проанализированы требования к приложению, проведен обзор существующих методов рефакторинга, описаны их достоинства и недостатки. Были созданы рекомендации к написанию кода. Для тестирования и отладки были разработаны модульные тесты. Были произведены работы по подготовке к проведению рефакторинга, а затем и был произведен рефакторинг программы автосопровождения.

После проведения работ по рефакторингу, количество методов значительно уменьшилось. В виду того, что было выявлено множество кусков алгоритмов, которые на данный момент потеряли свою актуальность. Алгоритмы ассоциации отметок, определения кругоответа и другие, существовали в проекте, но не использовались. Таким образом, в проекте уменьшилось количество функций с 160 до 126 методов. Размеры каждого не превышают 150–200 строк, ранее встречались функции размерами до десяти тысяч строк. Было выделено 13 классов, а общее количество файлов в проекте сократилось с 61 до 34.

Программа автосопровождения тестировалась при помощи модульных тестов во время проведения рефакторинга. Последним этапом тестирования был запуск на задокументированных реальных полетах летательных аппаратов.

В результате была создана новая версия программы автосопровождения летательных аппаратов, удовлетворяющая всем указанным требованиям. Таким образом, все поставленные задачи были успешно выполнены.

Новая версия программы автосопровождения летаельных аппаратов внедрена в изделие. В настоящее время проводятся пусконаладочные работы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Зыль, С.Н. Операционная система реального времени QNX: от теории к практике / С.Н. Зыль. – СПб.: БХВ–Петербург, 2004. – 192 с.
2. Кёртен, Р. Введение в QNX/Neutrino 2: Руководство по программированию приложений реального времени в QNX Realtime Platform / Р. Кёртен; пер. с англ. А.Н. Алексеев. – СПб.: Петрополис, 2004. – 514 с.
3. Зыль, С.Н. QNX Momentics: основы применения / С.Н. Зыль. – СПб.: БХВ–Петербург, 2005. – 256 с.
4. Блэк, Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование / Р. Блэк. – М.: Лори, 2014. – 544 с.
5. Павловская, Т.А. C/C++. Программирование на языке высокого уровня / Т.А. Павловская. – СПб.: Питер, 2003. – 461 с.
6. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер. – СПб.: Питер, 2004. – 318 с.
7. Фаулер, М. Рефакторинг: улучшение существующего кода / М. Фаулер; пер. с англ. С. Маккавеева. – СПб.: Символ Плюс, 2003. – 432 с.
8. Брауде, Э. Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004. – 656 с.
9. Страуструп, Б. Язык программирования C++. Специальное издание / Б. Страуструп; пер. с англ. Н.Н. Мартынова. – М.: Бином, 2011. – 1136 с.
10. Александровский, А. Современное проектирование на C++ / А.Александровский; пер. с англ. Д.А.Клюшина. – М.: Вильямс, 2002. – 326 с.
11. Паппас, К.Х. Отладка в C++. Руководство для разработчиков / К.Х. Паппас, У.Х. Мюррей. – М.: Бином, 2006. – 512 с.
12. Калбертсон, Р. Быстрое тестирование / Роберт Калбертсон, Крис Браун, Гэри Кобб. – М.: Вильямс, 2002. – 383 с.
13. Мартин, Р. Чистый код: создание, анализ и рефакторинг / Р. Мартин; пер. с англ. Е. Матвеев. – СПб.: Питер, 2018. – 464 с.
14. Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл; пер. с англ. – М.: Издательско-торговый дом Русская редакция; СПб.: Питер, 2005. – 896 стр.
15. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – М.: ДМК Пресс, 2008. – 496 с.
16. Лишнер, Р. C++. Справочник. Полное руководство по языку и стандартной библиотеке / Р. Лишнер; пер. с англ. А. Конев. – М.: Питер, 2005. – 906 с.
17. Шилдт, Г. Полный справочник по C++ / Г. Шилдт; пер. с англ. Д.А. Клюшина. – 4-е изд. – М.: Вильямс, 2006. – 800 с.
18. ГОСТ 34.602–89. Техническое задание на создание автоматизированной системы. – М.: Стандартинформ, 2006. – 17 с.
19. ГОСТ 19.504–79. Руководство программиста. Требования к содержанию и оформлению. – М.: Стандартинформ, 2010. – 2 с.

20. ГОСТ 19.402–78. Описание программы. – М.: Изд-во стандартов, 2000. – 49 с.

ПРИЛОЖЕНИЕ 1
ОПИСАНИЕ ПРОГРАММЫ

Работы: «Программа автосопровождения летательных аппаратов»

1. Общие сведения

Название программного продукта – «Программа автосопровождения летательных аппаратов». Программа написана на языке программирования C++ и функционирует под управлением операционной системой QNX. Для возможности функционирования программе требуется программное обеспечения изделия.

2. Функциональное назначение

Данная программа предназначена для организации приема входных отметок, формирования треков сопровождения и последующей выдачи потребителям.

Она решает проблему автоматизации обмена информацией, а также формирования треков летательных аппаратов, корректировку полетной информации, фильтрации отраженных сигналов и многими другими.

3. Описание логической структуры

Программа состоит из набора модулей, которые реализуют определенную часть функций и выполняют следующие действия:

- производят начальную загрузку программы;
- организовывают подключение к разделяемой памяти и очередям;
- организовывают формирование отладочной информации;
- производят прием и расшифровку входной информации;
- производят отождесвтеление координатных отметок;
- производят формирование треков сопровождения;
- производят сопровождение треков;
- организовывают формирование и выдачу информации потребителям.

Общий алгоритм работы программы можно представить в виде блок-схемы, изображенной на рисунке П1.1.

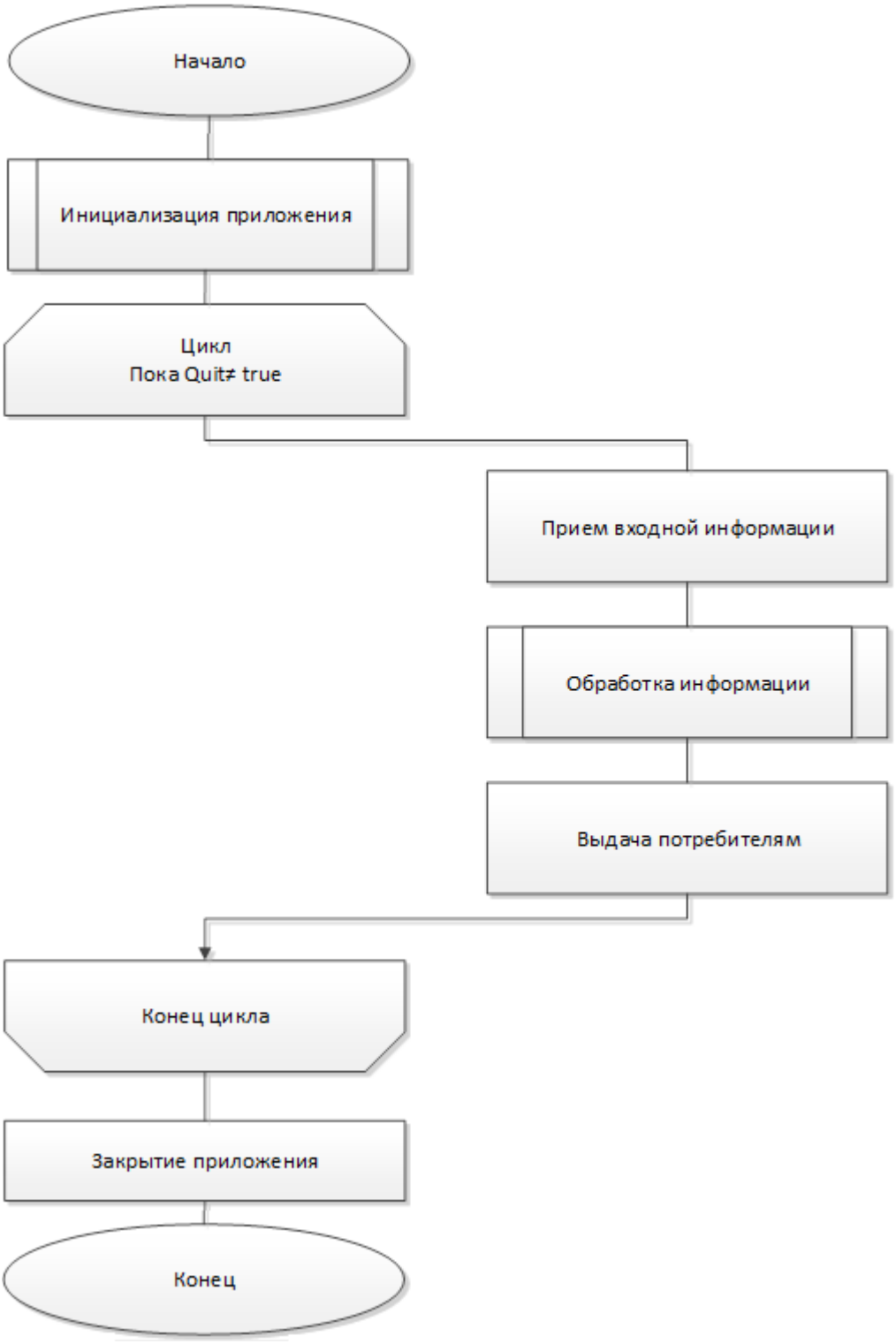

Рисунок П1.1 – Общий алгоритм работы программы

Функциональность среды определяется входящими в систему модулями, которые обеспечивают определенные функции для работы с веб-сайтами.

В рамках дипломного проекта разработаны следующие модули:

- модуль подключения к отладочному файлу и организации записи в него;
- модуль для организации и хранения подключения к разделяемой памяти;

- модуль для организации и хранения подключения к очередям;
- модуль для организации выдачи потребителям;
- модуль для организации фильтрации входной информации;
- модуль для организации поиска участков отраженных сигналов;
- модуль для организации подключения и обновления информации в сервисном окне;
- модуль для раскодирования входной информации;
- модуль для организации сопровождения треков;
- модуль для организации резервирования воздушной обстановки;
- модуль для организации отождествления отметок с треками сопровождения.

4. Используемые технические средства

Для обеспечения функционирования среды требуются следующие технические средства:

- IBM PC-совместимый компьютер;
- цветной монитор с диагональю не менее 15”;
- клавиатура;
- мышь.

Необходимые программно-аппаратные ресурсы представлены в таблице П1.1.

Таблица П1.1

Программно–аппаратная поддержка

Составляющие	Требования
Операционная система	QNX 6.5
Опреративная память, Гб	2
Пространство на жестком диске, Мб	100
Рабочая частота ЦПУ, ГГц	2

5. Входные и выходные данные

Входными данными для приложения является радиолокационная информация, состаящая из координатных отметок.

На выходе программа формирует сообщения потребилетям, содержащие информацию о треках сопровождения.

ПРИЛОЖЕНИЕ 2
ТЕКСТ ПРОГРАММЫ

Работы: «Программа автосопровождения летательных аппаратов»

Заголовочный файл ConnectSharedMemory.h

```
#ifndef Connect_SharedMemory_H_
#define Connect_SharedMemory_H_

#include <iostream>

#include "../../../share/map_ac_s.h"
#include "../../../share/map_pro1.h"
#include "../../../share/map_dnvrl.h"
#include "../../../../share/map_main.h"
#include "../../../../share/map_reg.h"
#include "../../../../share/map_karta.h"

#include "ConnectDebugLog.h"

class connectSharedMemory {
    public:
      static connectSharedMemory& GetConnect(){
            static connectSharedMemory object;
            return object;
      }

      PTRMAINSTRUCT   MapMain;
      MapMemAcS*        MapAcS;
      PTRREGSTRUCT      MapReg;
      PTRKARTASTRUCT    MapKarta;
      PTRPRO1STRUCT     MapPro;
      PTRDNVRLSTRUCT    MapDnvrl;

    private:
      connectSharedMemory();
      ~connectSharedMemory(){};

      connectSharedMemory( const connectSharedMemory& );
      connectSharedMemory& operator = ( connectSharedMemory& );

      void connectToMapMain();
      void connectToMapAcS();
      void connectToMapReg();
      void connectToMapPro();
      void connectToMapKarta();
      void connectToMapDnvrl();
};
#endif
```

Исполняемый файл ConnectSharedMemory.cpp

```
#include "ConnectSharedMemory.h"

connectSharedMemory::connectSharedMemory(){
    log_printff( DebugLog, "connectSharedMemory -> starting connected \n");
    connectToMapMain();
    connectToMapAcS();
    connectToMapReg();
    connectToMapPro();
    connectToMapKarta();
    connectToMapDnvrl();
    log_printff( DebugLog, "connectSharedMemory -> successfully connect ALL \n");
}

void connectSharedMemory::connectToMapMain(){
```

```
    if( !OpenSharedMem( "/map_main", O_RDWR, PROT_READ | PROT_WRITE,
                        ( void ** )&MapMain, sizeof( MAINSTRUCT ), NULL ) ) {
      log_printff( DebugLog, "connectSharedMemory--->
ERROR ! can't connect to MapMain \n");
      exit( -1 );
    }
    log_printff( DebugLog, "connectSharedMemory---> successfully connect to MapMain
    \n");
}
void connectSharedMemory::connectToMapAcS(){
    if( !OpenSharedMem( "/map_ac_s", O_RDWR, PROT_READ | PROT_WRITE,
                        ( void ** )&MapAcS, sizeof( ACSSHARESTRUCT ), NULL ) ) {
      log_printff( DebugLog, "connectSharedMemory--->
ERROR ! can't connect to MapAcS \n");
      exit( -1 );
    }
    log_printff( DebugLog, "connectSharedMemory--->
successfully connect to MapAcS \n");
}
void connectSharedMemory::connectToMapReg(){
    if( !OpenSharedMem( "/map_reg", O_RDONLY, PROT_READ,
                        ( void ** )&MapReg, sizeof( REGSHARESTRUCT ), NULL ) ) {
      log_printff( DebugLog, "connectSharedMemory--->
ERROR ! can't connect to MapReg \n");
      exit( -1 );
    }
    log_printff( DebugLog, "connectSharedMemory--->
successfully connect to MapReg \n");
}
void connectSharedMemory::connectToMapPro(){
    if( !OpenSharedMem( "/map_pro1", O_RDONLY, PROT_READ,
                          ( void ** )&MapPro, sizeof( PRO1SHARESTRUCT ), NULL ) ) {
      log_printff( DebugLog, "connectSharedMemory--->
ERROR ! can't connect to MapPro \n");
      exit( -1 );
    }
    log_printff( DebugLog, "connectSharedMemory--->
successfully connect to MapPro \n");
}
void connectSharedMemory::connectToMapKarta(){
    if( !OpenSharedMem( "/map_karta", O_RDONLY, PROT_READ,
                        ( void ** )&MapKarta, sizeof( KARTASTRUCT ), NULL ) ) {
      log_printff( DebugLog, "connectSharedMemory--->
ERROR ! can't connect to MapKarta \n");
      exit( -1 );
    }
    log_printff( DebugLog, "connectSharedMemory--->
successfully connect to MapKarta \n");
}
void connectSharedMemory::connectToMapDnvrl(){
    if( !OpenSharedMem( "/map_dnvrl", O_RDONLY, PROT_READ,
                        ( void ** )&MapDnvrl, sizeof( DNVRLSHARESTRUCT ), NULL ) ) {
      log_printff( DebugLog, "connectSharedMemory--->
ERROR ! can't connect to MapDnvrl \n");
      exit( -1 );
    }
    log_printff( DebugLog, "connectSharedMemory--->
successfully connect to MapDnvrl \n");
}
```

Заголовочный файл StageTracking.h

```
#ifndef ETAP_SOPR_H_
#define ETAP_SOPR_H_

#include "AzimuthCorrection.h"
#include "Filtration.h"
#include "SlidingBuffer.h"

#include "Connect/Connect.h"
#include "Connect/ConnectDebugLog.h"

#include "Configuration/ConfigurationFileReader.h"

class StageTracking {
  public:
        //    Запуск этапа сопровождения
        void StartingProcess (  MapMemAcS::_tr*   track,
        //    Запуск этапа сопровождения при пропуске КО
        void StartingProcess (  MapMemAcS::_tr*   track );
        //    Сглаживание  координат трека
        void SmoothedAzimuthAndRange( MapMemAcS::_tr*   track,
                                      unionMark*        mark );
        //    Экстраполяции координат трека
        void ExtrapolationCoordinates( MapMemAcS::_tr* track );
        //    Алгоритм сопровождения высоты
        void ProccessingHeight( MapMemAcS::_tr*   track,
                                unionMark*        mark  );
        //    Алгоритм сопровождения высоты при пропуске КО
        void ProccessingHeight( MapMemAcS::_tr*   track );
        //    Обновление инфорамции в треке
        void MarkToTrack( MapMemAcS::_tr*   track,
                          unionMark*        mark  );
        //    Инициализация стади сопровождения
        void Initialization( ConfigurationParameters*  _configParameters );
        StageTracking() {};
        ~StageTracking() {};
  private:
        AzimuthCorrection            azimuthCorrection;
        Filtration                   filtration;
        SlideBuffer                  slideBuffer;
        ReflectorArea                reflectorArea;
        ConfigurationParameters*     configParameters;
};
#endif
```

Заголовочный файл StageDecoding.h

```
#ifndef STAGE_DECODING_H_
#define STAGE_DECODING_H_

#include "StageIdentification.h"

#include "Connect/ConnectDebugLog.h"
#include "Connect/Connect.h"

#include "share/struct_pro_s.h"

#include "../../../lib/msg.h"
#include "../../../share/map_main.h"
#include "../../../share/struct_reg_s.h"

#include <iostream.h>

namespace constStageDecoding {
  static const double     CMR_HEIGHT_25F    =     25.0;
```

```
   static const double     CMR_METR_FUT      =     3.28;
};

class StageDecoding {
   public:
         StageDecoding(){};
         ~StageDecoding(){};

         void Initialization( ConfigurationParameters*  _configParameters );
         void decoding_message( Msg *ptr );
         void decoding_message_11();
         void decoding_message_4_5();
         void decoding_message_20_21();
         void decoding_message_KMA();

         void decodeHeight(unsigned short code, unionMark* mark);

         unsigned short transcodingWords( unsigned short code,int numbermask );
         unsigned short heightGilheimGray( unsigned short code );

         int getCountMsg()             { return rcvMsgCount; };
         int getCountOwerallMsg()      { return rcvOwerallMsgCount; };
         int getCountShortMsg()        { return rcvShortMsgCount; };
         int getCountLongMsg()         { return rcvLongMsgCount; };
         int getCountKmaMsg()          { return rcvKmaMsgCount; };

   private :
         char  *pointerCurrentPosition;

         int         rcvMsgCount,
                     rcvOwerallMsgCount,
                     rcvShortMsgCount,
                     rcvLongMsgCount,
                     rcvKmaMsgCount;

         StageIdentification          stageIdentification;
};
#endif
```

Заголовочный файл StageIdentification.h

```
#ifndef Stage_Identification_H_
#define Stage_Identification_H_

/*--------------------------------------------------------------------------*/
/*                         Этап отождествления отметок
/*--------------------------------------------------------------------------*/

#include "Asterix.h"
#include "SlidingBuffer.h"
#include "StageTracking.h"

#include "share/AvInSector.h"

#include "Connect/ConnectDebugLog.h"
#include "Connect/Connect.h"

#include <iostream.h>
#include <vector>
#include <algorithm>
#include <list>


class StageIdentification {
```

```
  public:
        StageIdentification(){};
        ~StageIdentification(){};

        void  Initialization( ConfigurationParameters*  _configParameters );
        void  startingProcess ( int numberKMA );
        void  ProcessingTrackWithoutKO( int numSektor );
        void  AcceleratedDispatch( int numSektor );
        int   CreateTrack( unionMark* mark, unionNtt* ntt1);
        double      deltaAzimuthWithSign( double az1, double az2 );
        bool  isFirstPrimaryOrFirstSpare();

  private :
        ConfigurationParameters*  configParameters;

        // Этап сопровождения
        StageTracking                       stageTracking;
        // Организация выдачи по протоколу Астерикс
        Asterix                             protokolASTERIX;
        SlideBuffer                         slideBuffer;

};
#endif
```

Заголовочный файл ServiceInfo.h

```
#ifndef SERVICE_INFO_H_
#define SERVICE_INFO_H_

#include "Connect/ConnectDebugLog.h"
#include "Connect/Connect.h"
#include "StageDecoding.h"
#include "../../../lib/service_info.h"
#include "../../../lib/ctl_io.h"
class ServiceInfo{

  public:
        ServiceInfo();
        ~ServiceInfo(){};

        void Initialization( StageDecoding* _stageDecoding );
        void Update();

  private:
        int                     serviceCoid;
        PTRSERVICEINFO    PtrService;
        StageDecoding*    stageDecoding;
        int                     CounterHealth;
};
#endif
```

Заголовочный файл Asterix.h

```
#ifndef _INC_ASTERIX_FILE__H_
#define _INC_ASTERIX_FILE__H_

#include <iostream>
#include <fstream>                    // ofstream
#include <cmath>                      // M_P
#include "share/Asterix.h"            // Все структуры выходных сообщений
#include "../../share/map_ac_s.h"     // MapMemAcS SMES
#include "../../../share/msgids.h"    // mstSrcSlave mstSrcMaster
#include "../../../share/macro.h"     // csMaster
#include "../../../share/map_reg.h"   // REGSTRUCT
#include "../../../share/map_main.h"
```

```
#include "Configuration/ConfigurationParameters.h"
#include "Connect/ConnectDebugLog.h"
#include "Connect/Connect.h"

namespace constAsterix {
   static const double     CMR_TIME  = 1.0f / 128.0f;
   static const double     CMR_H     = 7.62;
   static const double     CMR_A     = 360.0 / 65536;
   static const double     CMR_D     = 7.234375;

   static const double     BYTE_SYNX  =  0x10;
   static const double     BYTE_HOME  =  0x02;
   static const double     BYTE_END   =  0x03;

   static const double     TYPE_KAT_034 = 34;
   static const double     TYPE_KAT_048 = 48;
   static const double     TYPE_KAT_253 = 253;

   static const double     SAC          = 0x50;
   static const double     TYPE_SEVER   = 0x01;
   static const double     TYPE_SEKTOR  = 0x02;

   static const int              IRQ_IN_VO  = 5;   //Приоритет выдачи в очередь
};

class Asterix {

   public :
         Asterix(){};
         ~Asterix(){};
         void Initialization(    ConfigurationParameters*  configParameters);

         // Формирование и выдача кадра по протоколу Астерикс сообщения TRACK
         void codingTRACK (      int _tracknumber,
                                 MapMemAcS::_tr* _track );
         // Формирование и выдача кадра по протоколу Астерикс сообщения КМА
         void codingKMA ( int _tracknumber );
         // Формирование и выдача кадра по протоколу Астерикс сообщения СЕВЕР
         void codingNorth ( void );
         // Формирование и выдача кадра по протоколу Астерикс сообщения 253
         void codingMessage253 ( void );
   private:
         // Необходимые данные для корректной работы
         MapMemAcS::_tr* track;
         ConfigurationParameters* configParameters;
         // Номер трека
         int   track_number;
         // Выходное сообщение
         unsigned char     buffer_outputMessage[1024];
         // Длина выходного сообщения
         int               length_outputMessage;
         // Номер борта
         int   numberbort ;
         int   countTrackInSector ;

         // Прерывание формирование кадра по протоколу Астерикс сообщения TRACK
         bool isExitCodingTRACK ( void );
         // Формирование кадра по протоколу Астерикс сообщения TRACK
         void OrganizationMessageTRACK ( void );
         // Формирование кадра по протоколу Астерикс сообщения КМА
```

```
        void OrganizationMessageKMA ( void );
        // Формирование кадра по протоколу Астерикс сообщения СЕВЕР
        void OrganizationMessageNorth ( void );
        // Формирование кадра по протоколу Астерикс сообщения 253
        void OrganizationMessage253 ( void );
        // Вывод в консоль сформированного сообщения
        void printf_DebugInfo( void );
        // Повторение байта синхронизации в кадре по протоколу Астерикс
        void synchronizationbyteRepeating( void );
        // Выдача кадра по протоколу Астерикс в очередь QAST
        void write_messageAsterix( void );
        // Есть источник данных времени
        int isHasGpsTime( void );
        // Исправен ли источник времени
        int   isGpsTimeSourceCorrect( void );
        // Получить кол-во секунд от начала суток от GPS
        double getGpsTimeInSeconds( void );

        void creature_cadreHome ( unsigned char category );
        void creature_cadreEnd (  unsigned short checkSum );
        void creature_profileCategory_4 ( void );
        void creature_profileCategory_3 ( void );
        void creature_sourceIdentifier ( void );
        void creature_timesofDay( void );
        void creature_messageDescriptorFirst( void );
        void creature_messageDescriptorSecond( void );
        void creature_inclinedCoord( void );
        void creature_codeInOctal( int _numberbort );
        void creature_heightInBinary( void );
        void creature_trackNumber( int _numbertrack );
        void creature_trackStatus( void );
        void creature_codeInOctalRepresentationFirst( void );
        void creature_codeInOctalRepresentationSecond( void );
        void creature_dataSource(    unsigned short _numberSector );
        void creature_stateOfTheSystem(    void );
        void creature_profileCategory_253( void );
        void creature_typeMessage(   unsigned char  _typeMessage   );

        void creature_dateStatusRLS ( void );
        void creature_numberSector( unsigned short _numberSector );
        // Адрес воздушного судна (I048/220)
        void creature_aircraftAddress( void );
        // Идентификация воздушного судна (I048/240)
        void creature_identificationOfAircraft( void );
        // Данные режима S (I048/240)
        void creature_dataModeS( void );
        // Возможности связи/СПС/Статус полета (I048/230)
        void creature_communicationCapabilities_SPS_FlightStatus( void );
        // Информационное сообщение SPS (I048/260)
        void creature_messageSPS ( void );
        unsigned short KS_CRC( unsigned char *buffer, unsigned short size );
};
#endif
```

Заголовочный файл AzimuthCorrection.h

```
#ifndef AZIMUTH_CORRECTION_H_
#define AZIMUTH_CORRECTION_H_

#include "../../share/map_ac_s.h"
#include "../../share/struct_ac_s.h"

#include "ReflectorArea.h"
```

```
#include "SlidingBuffer.h"

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <math.h>
#include <fcntl.h>
#include <ctype.h>


class AzimuthCorrection {
  private :
        ReflectorArea* reflectorArea;
  public:
        AzimuthCorrection(){};
        ~AzimuthCorrection(){};
        void initialization( ReflectorArea* _reflectorArea);

        double meanMovement( MapMemAcS::_tr * track );
        double distanceBetweenTrackAndMark(       MapMemAcS::_tr * track,
                                                  double markx,
                                                  double marky );
        double distanceBetweenTrackAndMarkInd(    MapMemAcS::_tr * track,
                                                  unionMark * mark );
        double getStrobeForLinearMovementManeuver( MapMemAcS::_tr * track );
        double getStrobeForLinearMovement( MapMemAcS::_tr * track );

        int isReadyForAzimuthCorrection( MapMemAcS::_tr * track );
        int isMovementError( MapMemAcS::_tr * track, unionMark * mark);
        int isMovementErrorForManeuver(     MapMemAcS::_tr * track,
                                            unionMark * mark );
        int isTotalMovementError(           MapMemAcS::_tr * track,
                                            unionMark * mark,
                                            double coeff );
        int isLinearMovement(   MapMemAcS::_tr * track,
                                unionMark * mark );
        int isLinearMovementForManeuver(    MapMemAcS::_tr * track,
                                            unionMark * mark );
        int isNeedAzimuthCorrection( MapMemAcS::_tr * track,
                                     unionMark * mark );
        void clearLinearCounterForTrack( MapMemAcS::_tr * track );
        void incLinearCounterForTrack( MapMemAcS::_tr * track );
        unsigned short getLinearCounterForTrack( MapMemAcS::_tr * track );
        void doMarkAzimuthCorrection( MapMemAcS::_tr * track,
                                      unionMark * mark );
        int isRangeDifferenceConstant(      MapMemAcS::_tr * track,
                                            unionMark * mark );
        double distanceBetweenTrackEstAndMark(    MapMemAcS::_tr * track,
                                                  unionMark * mark );

};
#endif
```

Заголовочный файл Filtration.h

```
#ifndef FILTRATION_H_
#define FILTRATION_H_

// ----------------------------------------------------------------------
//                        Включаемые файлы
// ----------------------------------------------------------------------

#include "ReflectorArea.h"
```

```
#include <math.h>
#include "../../share/struct_ac_s.h"
// ---------------------------------------------------------------------------
//                          Константные значения
// ---------------------------------------------------------------------------

namespace constFiltration {
   static const double SIGA = 0.3;
   static const double SIGD = 100.0;
}

// ---------------------------------------------------------------------------
//                          Прототипы функций
// ---------------------------------------------------------------------------
class Filtration {
   private :
         ReflectorArea* reflectorArea;
   public:
         Filtration(){};
         ~Filtration(){};
         void initialization( ReflectorArea* _reflectorArea);
         //    Альфа-Бета фильтрация координатной информации
         void abFilter ( MapMemAcS::_tr* track , double t_obz );
         //    Кооректировка альфа-бета коэфф.
         void selectABForFiltering( MapMemAcS::_tr* track );
         //    Функция проверяющая завершение маневра.
         int  checkEndManeuverAndSelectAB(   MapMemAcS::_tr* track,
                                             unionMark* mark );
         //    Функция проверяющая начинается ли маневр?
         int  checkStartManeuverAndSelectAB(MapMemAcS::_tr* track,
                                             unionMark* mark ,
                                             int isManMove);
};
#endif
```

Модульный тест Filtration.h

```
#include "../Filtration.h"
#include "gtest/gtest.h"

// ---------------------------------------------------------------------------
//  1. Тестирование Альфа-Бета фильтрация координатной информации
// ---------------------------------------------------------------------------
TEST( abFilterTesting , smoothedCoord_0_5 ) {

   Map_Of_Mem_Ac::_tr  track;
   track.markX = track.markY = 2500;
   track.x = track.y = 1000;
   track.alfa = 0.5;
   abFilter( &track , 100 );
   EXPECT_EQ(1750, track.xs );
   EXPECT_EQ(1750, track.ys );
}
TEST( abFilterTesting , smoothedCoord_0_2 ) {

   Map_Of_Mem_Ac::_tr  track;
   track.markX = 500;
   track.markY = 1000;
   track.x = 1000;
   track.y = 500;
   track.alfa = 0.2;
   abFilter( &track , 100 );
   EXPECT_EQ( 900, track.xs );
   EXPECT_EQ( 600, track.ys );
```

```
}
TEST( abFilterTesting , smoothedCoord_0_8 ) {

   Map_Of_Mem_Ac::_tr  track;
   track.markX = 9500;
   track.markY = 2000;
   track.x = 10000;
   track.y = 1000;
   track.alfa = 0.8;
   abFilter( &track , 100 );
   EXPECT_EQ( 9600, track.xs );
   EXPECT_EQ( 1800, track.ys );
}
TEST( abFilterTesting , smoothedAzimuth ) {

   Map_Of_Mem_Ac::_tr  track;
   track.markX = 1500;
   track.markY = 2000;
   track.x = 700;
   track.y = 1000;
   track.alfa = 0.3;
   abFilter( &track , 100 );
   EXPECT_FLOAT_EQ( 0.94 , round ( track.as * 100 ) / 100 );
}
TEST( abFilterTesting , smoothedDalnost ) {

   Map_Of_Mem_Ac::_tr  track;
   track.markX = 1500;
   track.markY = 2000;
   track.x = 700;
   track.y = 1000;
   track.alfa = 0.3;
   abFilter( &track , 100 );
   EXPECT_FLOAT_EQ( 1604.24 , round ( track.ds * 100 ) / 100 );
}
TEST( abFilterTesting , smoothedVelocity ) {

   Map_Of_Mem_Ac::_tr  track;
   track.markX = 1500;
   track.markY = 2000;
   track.x = 700;
   track.y = 1000;
   track.alfa = 0.3;
   track.beta = 0.2;
   track.wx = track.wy = 0;
   abFilter( &track , 10 );
   EXPECT_EQ( 16, track.wx );
   EXPECT_EQ( 20, track.wy );
}
// ------------------------------------------------------------------------
//  2. Тестирование процедуры кооректировки альфа-бета коэфф.
// ------------------------------------------------------------------------
TEST( selectABForFilteringTesting , zeroCount ) {
   Map_Of_Mem_Ac::_tr  track;
   track.cnt = 0;
   selectABForFiltering( &track );
   EXPECT_FLOAT_EQ( 0.46f , round( track.alfa * 100) / 100 );
   EXPECT_FLOAT_EQ( 0.11f , round( track.beta * 100) / 100 );
}
TEST( selectABForFilteringTesting , threeCount ) {
   Map_Of_Mem_Ac::_tr  track;
   track.cnt = 3;
```

```
   selectABForFiltering( &track );
   EXPECT_FLOAT_EQ( 0.83f, round( track.alfa * 100) / 100 );
   EXPECT_FLOAT_EQ( 0.5f , round( track.beta * 100) / 100 );

}
TEST( selectABForFilteringTesting , _mtNoManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   track.cnt = 10;
   track.maneuver_type = mtNoManeuver;
   selectABForFiltering( &track );
   EXPECT_FLOAT_EQ( 0.3f , round( track.alfa * 100) / 100 );
   EXPECT_FLOAT_EQ( 0.05f , round( track.beta * 100) / 100 );
}
TEST( selectABForFilteringTesting , _mtPotentialManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   track.cnt = 10;
   track.maneuver_type = mtPotentialManeuver;
   selectABForFiltering( &track );
   EXPECT_FLOAT_EQ( 0.3f , round( track.alfa * 100) / 100 );
   EXPECT_FLOAT_EQ( 0.05f , round( track.beta * 100) / 100 );
}
TEST( selectABForFilteringTesting , _mtManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   track.cnt = 10;
   track.maneuver_type = mtManeuver;
   selectABForFiltering( &track );
   EXPECT_FLOAT_EQ( 0.6f , round( track.alfa * 100) / 100 );
   EXPECT_FLOAT_EQ( 0.4f , round( track.beta * 100) / 100 );
}
TEST( selectABForFilteringTesting , _mtHiManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   track.cnt = 10;
   track.maneuver_type = mtHiManeuver;
   selectABForFiltering( &track );
   EXPECT_FLOAT_EQ( 0.75f , round( track.alfa * 100) / 100 );
   EXPECT_FLOAT_EQ( 0.45f , round( track.beta * 100) / 100 );
}
// ------------------------------------------------------------------------
//  3. Функция проверяющая завершение маневра.
//     Корректирует альфа-бета коэфициенты
// ------------------------------------------------------------------------
TEST( checkEndManeuverAndSelectABTesting , zeroCount ) {
   Map_Of_Mem_Ac::_tr  track;
   Map_Of_Mem_Ac::_nak mark;
   mark.a = 140;
   mark.d = 15000;
   track.a = 141;
   track.d = 16000;
   track.cnt = 0;
   EXPECT_EQ ( 0 , checkEndManeuverAndSelectAB( &track, &mark) );
}

TEST( checkEndManeuverAndSelectABTesting , notStrobAzimut ) {
   Map_Of_Mem_Ac::_tr  track;
   Map_Of_Mem_Ac::_nak mark;
   mark.a = 140;
   mark.d = 15000;
   track.a = 200;
   track.d = 16000;
   track.cnt = 10;
   EXPECT_EQ ( 0 , checkEndManeuverAndSelectAB( &track, &mark) );
}
```

```
TEST( checkEndManeuverAndSelectABTesting , notStrobDalnost ) {
   Map_Of_Mem_Ac::_tr  track;
   Map_Of_Mem_Ac::_nak mark;
   mark.a = 140;
   mark.d = 15000;
   track.a = 141;
   track.d = 25000;
   track.cnt = 10;
   EXPECT_EQ ( 0 , checkEndManeuverAndSelectAB( &track, &mark) );
}
TEST( checkEndManeuverAndSelectABTesting , _mtHiManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   Map_Of_Mem_Ac::_nak mark;
   mark.a = 100;
   mark.d = 15000;
   track.a = 100;
   track.d = 15000;
   track.cnt = 10;
   track.maneuver_type = mtHiManeuver;
   EXPECT_EQ ( 1 , checkEndManeuverAndSelectAB( &track, &mark) );
   EXPECT_EQ ( mtManeuver , track.maneuver_type );
   EXPECT_FLOAT_EQ( 0.6f , round( track.alfa * 10) / 10 );
   EXPECT_FLOAT_EQ( 0.4f , round( track.beta * 10) / 10 );
}
TEST( checkEndManeuverAndSelectABTesting , _mtManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   Map_Of_Mem_Ac::_nak mark;
   mark.a = 1;
   mark.d = 15000;
   track.a = 1;
   track.d = 15000;
   track.cnt = 10;
   track.maneuver_type = mtManeuver;
   EXPECT_EQ ( 1 , checkEndManeuverAndSelectAB( &track, &mark) );
   EXPECT_EQ ( mtNoManeuver , track.maneuver_type );
   EXPECT_FLOAT_EQ( 0.6f , round( track.alfa * 100) / 100 );
   EXPECT_FLOAT_EQ( 0.2f , round( track.beta * 10) / 10 );
}
TEST( checkEndManeuverAndSelectABTesting , _mtPotentialManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   Map_Of_Mem_Ac::_nak mark;
   mark.a = 1;
   mark.d = 15000;
   track.a = 1;
   track.d = 15000;
   track.cnt = 10;
   track.maneuver_type = mtPotentialManeuver;
   EXPECT_EQ ( 1 , checkEndManeuverAndSelectAB( &track, &mark) );
   EXPECT_EQ ( mtNoManeuver , track.maneuver_type );
   EXPECT_FLOAT_EQ( 0.3f , round( track.alfa * 10 ) / 10 );
   EXPECT_FLOAT_EQ( 0.05f , round( track.beta * 100) / 100 );
}


// ------------------------------------------------------------------------
//  4. Функция проверяющая начинается ли маневр?
//     Корректирует альфа-бета коэфициенты
// ------------------------------------------------------------------------
TEST( checkStartManeuverAndSelectABTesting , _mtPotentialManeuver ) {
   Map_Of_Mem_Ac::_tr  track;
   Map_Of_Mem_Ac::_nak mark;
   track.cnt = 10;
```

```
      track.maneuver_type = mtPotentialManeuver;
      EXPECT_EQ ( 1 ,checkStartManeuverAndSelectAB ( &track, &mark, 0 ) );
      EXPECT_EQ ( mtManeuver , track.maneuver_type );
      EXPECT_FLOAT_EQ( 0.6f , round( track.alfa * 10 ) / 10 );
      EXPECT_FLOAT_EQ( 0.4f , round( track.beta * 10 ) / 10 );
   }
   TEST( checkStartManeuverAndSelectABTesting , _mtManeuver ) {
      Map_Of_Mem_Ac::_tr  track;
      Map_Of_Mem_Ac::_nak mark;
      mark.a = 1;
      mark.d = 15000;
      track.a = 100;
      track.d = 20000;
      track.cnt = 10;
      track.maneuver_type = mtManeuver;
      EXPECT_EQ ( 1 ,checkStartManeuverAndSelectAB ( &track, &mark, 0 ) );
      EXPECT_EQ ( mtHiManeuver , track.maneuver_type );
      EXPECT_FLOAT_EQ( 0.75f , round( track.alfa * 100 ) / 100 );
      EXPECT_FLOAT_EQ( 0.45f , round( track.beta * 100 ) / 100 );
   }
   TEST( checkStartManeuverAndSelectABTesting , _mtNoManeuver ) {
      Map_Of_Mem_Ac::_tr  track;
      Map_Of_Mem_Ac::_nak mark;
      mark.a = 1;
      mark.d = 15000;
      track.a = 1;
      track.d = 15000;
      track.cnt = 10;
      track.maneuver_type = mtNoManeuver;
      EXPECT_EQ ( 0 ,checkStartManeuverAndSelectAB ( &track, &mark, 0 ) );
      EXPECT_EQ ( mtNoManeuver , track.maneuver_type );
   }
   TEST( checkStartManeuverAndSelectABTesting , isManStrobeDalnost_mtNoManeuver )
{
      Map_Of_Mem_Ac::_tr  track;
      Map_Of_Mem_Ac::_nak mark;
      mark.a = 1.22;
      mark.d = 15000;
      track.a = 1.22;
      track.d = 15200;
      track.cnt = 10;
      track.maneuver_type = mtNoManeuver;
      EXPECT_EQ ( 1 ,checkStartManeuverAndSelectAB ( &track, &mark, 0 ) );
      EXPECT_EQ ( mtPotentialManeuver , track.maneuver_type );
   }

   TEST( checkStartManeuverAndSelectABTesting , isManStrobeAzimuth_mtNoManeuver )
{
      Map_Of_Mem_Ac::_tr  track;
      Map_Of_Mem_Ac::_nak mark;
      mark.a = 1.39;
      mark.d = 15000;
      track.a = 1.22;
      track.d = 15000;
      track.cnt = 10;
      track.maneuver_type = mtNoManeuver;
      EXPECT_EQ ( 1 ,checkStartManeuverAndSelectAB ( &track, &mark, 0 ) );
      EXPECT_EQ ( mtPotentialManeuver , track.maneuver_type );
   }
   TEST( checkStartManeuverAndSelectABTesting , isManMove_mtNoManeuver ) {
      Map_Of_Mem_Ac::_tr  track;
      Map_Of_Mem_Ac::_nak mark;
```

```
   mark.a = 1.56;
   mark.d = 19000;
   track.a = 1.57;
   track.d = 19000;
   track.cnt = 10;
   track.maneuver_type = mtNoManeuver;
   EXPECT_EQ ( 1 ,checkStartManeuverAndSelectAB ( &track, &mark, 1 ) );
   EXPECT_EQ ( mtPotentialManeuver , track.maneuver_type );
}
```

Модульный тест Asterix.h

```
#include "../../u_asterix_NEW.h"
#include "gtest/gtest.h"

//-----------------------------------------------------------------------------
//  1. Тестирование формирования начального кадра категорий
//-----------------------------------------------------------------------------
TEST( creature_cadreHome , isCategory34 ) {

   Asterix protokolASTERIX;

   protokolASTERIX.creature_cadreHome ( constAsterix::TYPE_KAT_034 );

   EXPECT_EQ(  sizeof( kadr_nome_ ), protokolASTERIX.length_outputMessage );

}
TEST( creature_cadreEnd , isControlSumNULL ) {

   Asterix protokolASTERIX;

   protokolASTERIX.creature_cadreEnd ( 0 );

   EXPECT_EQ(  sizeof( kadr_end_ ), protokolASTERIX.length_outputMessage );
}
TEST( creature_profileCategory_4 ,    isModeCodeTrue ) {

   Asterix protokolASTERIX;

   protokolASTERIX.creature_profileCategory_4 ( false );

   EXPECT_EQ(  sizeof( p_kat_048_4_ ), protokolASTERIX.length_outputMessage );
}
TEST( creature_profileCategory_2 ,    isDefault_ ) {

   Asterix protokolASTERIX;

   protokolASTERIX.creature_profileCategory_2 ( );

   EXPECT_EQ(  sizeof( p_kat_048_2_ ), protokolASTERIX.length_outputMessage );
}
TEST( creature_sourceIdentifier ,     isDefault_ ) {

         Asterix protokolASTERIX;

   protokolASTERIX.creature_sourceIdentifier( );

   EXPECT_EQ(  sizeof( isd_ ), protokolASTERIX.length_outputMessage );
}
TEST( creature_creature_timesofDay ,  isDefault_ ) {
```

```
   Asterix protokolASTERIX;

   protokolASTERIX.creature_timesofDay( );

   EXPECT_EQ(  sizeof( time_sut_ ), protokolASTERIX.length_outputMessage );
}
TEST( creature_messageDescriptorFirst,      isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_messageDescriptorFirst( );

   EXPECT_EQ(  sizeof( osc1_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_messageDescriptorSecond,     isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_messageDescriptorSecond();

   EXPECT_EQ(  sizeof( osc2_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_inclinedCoord,   isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_inclinedCoord();

   EXPECT_EQ(  sizeof( nk_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_codeInOctalRepresentation,   isZeroNumberBort ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_codeInOctalRepresentation( 0 );

   EXPECT_EQ(  sizeof( kr_3a_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_flightLevelInBinatuRepresentation, isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_flightLevelInBinatuRepresentation();

   EXPECT_EQ(  sizeof( h2_ ) , protokolASTERIX.length_outputMessage );
}

TEST( creature_trackNumber,     isZeroTrackNumber ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_trackNumber( 0 );

   EXPECT_EQ(  sizeof( ntr_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_trackStatus,     isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_trackStatus();

   EXPECT_EQ(  sizeof( str1_ ) , protokolASTERIX.length_outputMessage );
}

TEST( creature_codeInOctalRepresentationFirst,    isDefault_ ) {


   Asterix protokolASTERIX;
   protokolASTERIX.creature_codeInOctalRepresentationFirst();
```

```
   EXPECT_EQ(  sizeof( kr_k1_ ) , protokolASTERIX.length_outputMessage );
}

TEST( creature_codeInOctalRepresentationSecond,   isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_codeInOctalRepresentationSecond();

   EXPECT_EQ(  sizeof( kr_k2_ ) , protokolASTERIX.length_outputMessage );
}

TEST( creature_profileNonStandardDataElements,    isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_profileNonStandardDataElements();

   EXPECT_EQ(  sizeof( p_pol_ned_ ) , protokolASTERIX.length_outputMessage );
}

TEST( creature_dataAvaliability,isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_dataAvaliability();

   EXPECT_EQ(  sizeof( p_nd_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_numberBort,isZeroNumberBort) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_numberBort( 0 );

   EXPECT_EQ(  sizeof( nb_uvd_ ) , protokolASTERIX.length_outputMessage );
}

TEST( creature_heightAndfuel,   isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_heightAndfuel( );

   EXPECT_EQ(  sizeof( h_uvd_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_dataSource,isZeroNumberSector ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_dataSource( 0 );

   EXPECT_EQ(  sizeof( p_kat_034_ ) , protokolASTERIX.length_outputMessage );
}

TEST( creature_stateOfTheSystem,isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.creature_stateOfTheSystem( );

   EXPECT_EQ(  sizeof( conf_ctrl_s_ ) , protokolASTERIX.length_outputMessage );
}
TEST( creature_profileCategory_253,   isDefault_ ) {


   Asterix protokolASTERIX;
   protokolASTERIX.creature_profileCategory_253( );
```

```
      EXPECT_EQ(  sizeof( p_kat_253_ ) , protokolASTERIX.length_outputMessage );
    }
    TEST( creature_typeMessage,     isZeroTypeMessage) {

      Asterix protokolASTERIX;
      protokolASTERIX.creature_typeMessage( 0 );

      EXPECT_EQ(   sizeof( unsigned char ) , protokolASTERIX.length_outputMessage
);
    }
    TEST( creature_dateStatusRLS,   isDefault_ ) {

      Asterix protokolASTERIX;
      protokolASTERIX.creature_dateStatusRLS( );

      EXPECT_EQ(  sizeof( d_sos_rls_ ) , protokolASTERIX.length_outputMessage );
    }
    TEST( creature_numberSector,    isZeroNumberSector ) {

      Asterix protokolASTERIX;
      protokolASTERIX.creature_numberSector( 0 );

      EXPECT_EQ(  sizeof( num_s_ ) , protokolASTERIX.length_outputMessage );
    }

    TEST( creature_lengthDomesticChannel, isZeroLengthDomesticChannel ) {

      Asterix protokolASTERIX;
      protokolASTERIX.creature_lengthDomesticChannel( 0 );

      EXPECT_EQ(   sizeof( unsigned short ) , protokolASTERIX.length_outputMessage
);
    }

    TEST( OrganizationMessageNorth ,isDefault_ ) {

      Asterix protokolASTERIX;
      protokolASTERIX.OrganizationMessageNorth();

      EXPECT_EQ(  16 , protokolASTERIX.length_outputMessage );
    }
    TEST( OrganizationMessage253,   isDefault_ ) {

      Asterix protokolASTERIX;
      protokolASTERIX.OrganizationMessage253();

      EXPECT_EQ(  14 , protokolASTERIX.length_outputMessage );
    }
    TEST( OrganizationMessageKMA,   isDefault_ ) {


      Asterix protokolASTERIX;
      protokolASTERIX.OrganizationMessageKMA();

      EXPECT_EQ(  17 , protokolASTERIX.length_outputMessage );
    }
    TEST( OrganizationMessageTRACK ,isDefault_ ) {


      Asterix protokolASTERIX;
      protokolASTERIX.OrganizationMessageTRACK();
```

```
   EXPECT_EQ(  25 , protokolASTERIX.length_outputMessage );
}
TEST( TwoOrganizationMessageTRACK ,   isDefault_ ) {

   Asterix protokolASTERIX;
   protokolASTERIX.OrganizationMessageTRACK();
   protokolASTERIX.OrganizationMessageTRACK();

   EXPECT_EQ(  41 , protokolASTERIX.length_outputMessage );
}
```

Заголовочный файл ReflectorArea.h

```
#ifndef REFLECTOR_AREA_H_
#define REFLECTOR_AREA_H_


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fcntl.h>
#include <ctype.h>

#include "../../share/map_ac_s.h"
#include "share/SectorArea.h"

#define REFLECTOR_AREA_COUNT 3
#define  PI M_PI

class ReflectorArea {
   private:
         int reflectorAreasCount;
         SectorArea reflectorAreas[ REFLECTOR_AREA_COUNT ];
   public:
         ReflectorArea();
         ~ReflectorArea(){};
         int isTrackInReflectorArea( MapMemAcS::_tr* track );
         int isInReflectorArea( double azimuth, double range );
         void readReflectorAreas( const char* fileName );
         char *getFirstSymbol( char *s );
};
#endif
```

Заголовочный файл RliReserve.h

```
#ifndef RliReserve_H_
#define RliReserve_H_

#include "Connect/ConnectDebugLog.h"
#include "Connect/Connect.h"

#include "share/struct_rli_reserve.h"

#include "../../../share/msgids.h"

class RliReserve {
   public:
         void decodingMessage     ( Msg *ptr );
         void sendMessage         ( void );
         void organizationMessage ( void );
         void printfDebugInfo     ( void );
         void clearMessage        ( void );
```

```
        RliReserve();
        ~RliReserve(){};

  private:
        unsigned char     bufferOutputMessage[1024];
        int               lengthOutputMessage;
};
#endif
```