Обзор внедрения модельно-ориентированного проектирования в предприятиях аэрокосмической отрасли

Глоссарий

Трассируемость - доказательство связи различных единиц
Требования высокого уровня - требования к ПО, разработанные на основе анализа требований к системе, требований к безопасности и архитектуре системы
Требования низкого уровня - требования к ПО, полученные из требований высокого уровня, производных требований и проектных ограничений, которые можно непосредственно реализовать в исходном коде без привлечения дополнительной информации
МОП - модельно-ориентированное проектирование
Стандарт кодирования - набор правил, ограничивающий язык программирования и устанавливающий правила оформления исходного кода
Формальные методы - описательная символика (система обозначений) и аналитические методы, используемые для составления, разработки и обоснования математических моделей работы системы.
Регрессионный анализ - анализ влияния изменений на поведение системы
Верификация - оценка результатов некоторого процесса для гарантии его корректности и соответствия входным данным и стандартам, определенным для этого процесса
Валидация - процесс определения правильности и полноты требований.
Моделирование - процесс создания математической модели изделия или процесса с целью получения новых знаний
Симуляция - использование ранее созданных моделей с целью получения новых знаний или проектирования

1. Введение

Разработка бортового программного обеспечения ведется в соответствии со стандартом КТ-178С “Требования к программному обеспечению бортовой аппаратуры и систем при сертификации авиационной техники”. Классическая реализация предлагаемого процесса разработки достаточно трудоемка даже для простейших систем. Однако существует современная методология разработки, существенно ускоряющая этот процесс - **модельно-ориентированное проектирование**. Важно отметить, что данная методология успешно применяется в ведущих предприятиях авиапромышленного комплекса.

На протяжении всего времени существования авионики роль программного обеспечения в бортовых системах постоянно возрастала, а сами бортовые системы становились все более комплексными и сложными.
Это привело к тому, что бортовое программное обеспечение должно разрабатываться большими командами, должно быть учтено взаимовлияние аппаратного и программного обеспечения, а также различных подсистем. Неизбежно проекты стали сталкиваться со следующими трудностями:

- Зависимость от стендового ПО, симуляторов, физических прототипов
- Коммуникация как внутри команды разработчиков, так и между командами и предприятиями
- Непрозрачность связи между исходными требованиями и конечным результатом

- Лавинообразное нарастание сложности исходного кода
- Большие затраты на верификацию и валидацию систем
- Проблемы с квалификацией разнородных инструментов разработки

Ответом на эти вызовы может служить применение модельно-ориентированного проектирования, что и рассматривается в настоящей статье.
Необходимо также отметить, что методология модельно-ориентированного проектирования успешно применяется при разработке бортового радиоэлектронного оборудования (БРЭО) и авиационного оборудования (АО) и признана сертифицирующими органами по всему миру.

2. Модельно-ориентированное проектирование для разработки сложных систем в соответствии с КТ-178С

Применение модельно-ориентированного проектирования и системных моделей обеспечивает быструю разработку встраиваемых алгоритмов

Модельно-ориентированное проектирование (**далее, МОП**) – это современный подход к разработке больших, сложных и высоконадежных систем. Суть метода заключается в систематическом применении моделей на всех этапах жизненного цикла ПО, от проектирования до реализации. В ходе разработки, в соответствии с этим подходом, создается **системная модель**, включающая в себя модели как физической части (гидравлика, электрика и т.д.) так и самих алгоритмов. Такая модель используется для оценки взаимодействия алгоритма, физической части и внешней среды. Также тестирование и верификация алгоритмов проводятся на ранних стадиях проекта без использования физических прототипов.

Рисунок 1 Процесс МОП

Из полученных моделей алгоритмов генерируется исходный код как для ПЛИС, так и для микропроцессоров. Таким образом сокращаются затраты на этап кодирования, а сами алгоритмы являются независимыми от целевой платформы.

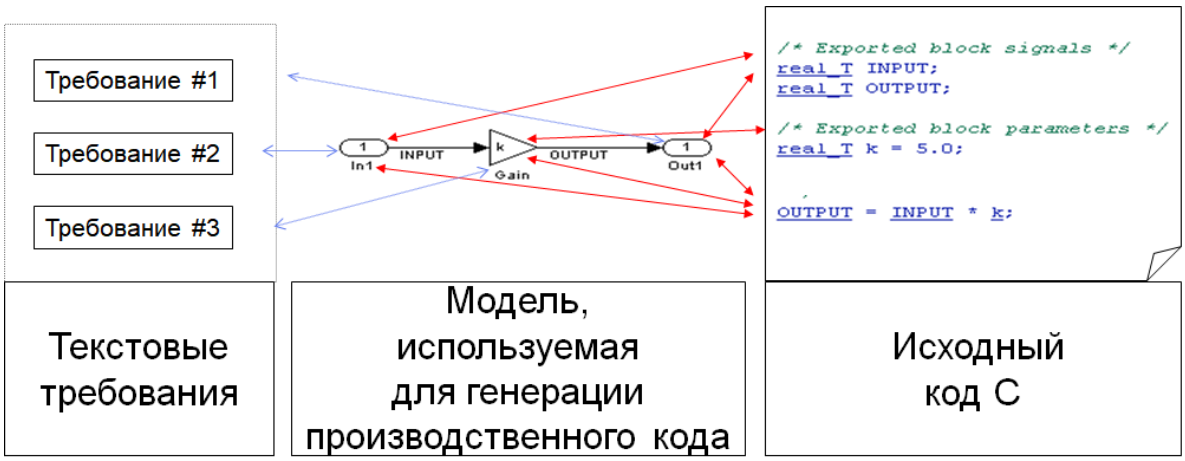

Рисунок 2 Связь требований кода и модели

Рисунок 3 Соотношение количества выявленных ошибок на стадиях проекта при традиционном подходе (красный) и при применении МОП (синий)

Эффекты применения модельно-ориентированного проектирования:

- Сокращение трудозатрат и времени разработки алгоритмов
- Сокращение трудозатрат и времени реализации алгоритмов на целевой микропроцессорной платформе, благодаря автоматической генерации С-кода
- Валидация требований к системе с помощью создания системной модели и проведения испытаний на ней
- Отладка и тестирование алгоритмов на уровне модели системы, что обеспечивает нахождение большинства ошибок разработки на стадии эскизного проектирования, а не на стадии тестирования прототипа устройства
- Прозрачность связи требований, модели, С-кода и тестовых сценариев, что является одним из условий соответствия отраслевым стандартам разработки встраиваемых систем

Основанием применения модельно-ориентированного проектирования в разработке бортового ПО являются следующие материалы:

- Руководство Р-4754 по процессам сертификации высокоинтегрированных сложных бортовых систем воздушных судов гражданской авиации
- Руководство Р-331, «Разработка и верификация на основе модели»

Руководство Р-4754 прямо указывает на применение системных моделей для разработки алгоритмов. Р-331 описывает процесс разработки по методологии модельно-ориентированного проектирования согласно КТ-178С. В следующих разделах дается

разъяснение по применению модельно-ориентированного проектирования на всех этапах жизненного цикла ПО и предлагаются лучшие практики его внедрения.

3. Модельно-ориентированное проектирование как процесс разработки ПО

Метод проектирования сверху-вниз и итеративная модель разработки обеспечивают обнаружение ошибок на ранних стадиях проекта

Процесс разработки ПО описывается т.н. моделями жизненного цикла. Выбор такой модели жизненного цикла разработки ПО - важный этап на начале разработки. В настоящее время устоявшейся практикой в отрасли является применение следующих моделей:

- Каскадная модель (Waterfall)
- Итерационная или спиральная модель (Spiral Lifecycle)
- Гибкая разработка (Agile)

Рассмотрение сильных и слабых сторон данных моделей выходит за рамки настоящей статьи, однако выбор модели должен быть продиктован маршрутом проектирования. Ниже будет приведен обзор маршрута проектирования, рекомендованного для МОП и рекомендована наиболее подходящая модель жизненного цикла.

Для построения маршрута проектирования необходимо иметь в виду, что:

- Требуется обеспечивать полную трассируемость между требованиями, тестовыми векторами, кодом и моделями
- Требуется проводить систематические валидацию и верификацию
- Так как разработка и функциональное тестирование ведется при помощи симуляций, тестовые вектора для подтверждения функциональной корректности системы разрабатываются на ранних этапах разработки.

С учетом этих требований маршрут проектирования можно представить в виде прохождения следующих этапов:

1. Минимальный прототип
2. Уточнение модели
3. Верификация модели
4. Генерация исходного кода
5. Верификация исходного кода

В ходе вышеприведенных процессов и мероприятий будут создаваться артефакты жизненного цикла, которые могут быть представлены сертифицирующему органу. Ниже приведены пояснения по каждому этапу.

Сначала разрабатывается минимальный прототип (*low-fidelity model*), отвечающий функциональным требованиям и достигается максимальное покрытие требований высокого уровня.

Затем прототип проходит несколько стадий уточнения. При этом созданные для минимального прототипа тестовые вектора применяются и к уточненным моделям. В ходе уточнения прототипа может оказаться, что требования высокого уровня были сформулированы либо недостаточно полно, либо неточно. В таком случае требования уточняются, что приводит к сдвигу верификации и валидации «влево», на этапы формирования требований и проектирования.

Таким образом, после нескольких итераций будет получена функционально корректная модель системы или компонента. После подтверждения функциональной корректности требуется показать соответствие модели стандартам моделирования, а также отсутствие ошибок проектирования.

После выполнения мероприятий по разработке модели осуществляется переход к реализации модели в коде. В отличие от предыдущих этапов, на данном этапе требуется определить целевую платформу для реализации. Выбор целевой платформы будет влиять на дальнейшие действия с моделью, например, может потребоваться перевод модели из арифметики с плавающей точкой в арифметику с фиксированной точкой. Такие модификации, очевидно, приведут к повторной верификации модели.
Финальным этапом процесса реализации будет генерация исходного кода для целевой платформы и его верификация.
Процессу разработки, описанному выше, соответствует **спиральная,** или **итерационная** модель жизненного цикла разработки ПО, а метод проектирования называется **метод “сверху-вниз”**.

4. Особенности проектирования систем в методологии МОП

Сочетание модельно-ориентированного проектирования и итеративной модели обеспечивают прозрачность разработки

4.1 Создание системной модели

Рассмотрим обобщенную задачу проектирования технической системы. Такая система состоит из компонентов, связанных между собой. В общем случае, каждый компонент будет обладать как входами, так и выходами, то есть интерфейсом. При применении метода проектирования “сверху-вниз” первыми этапами разработки будут следующие мероприятия:

- Определение компонентов системы
- Определение интерфейсов компонентов

После определения компонентов системы создается системная модель, содержащая все компоненты в виде подсистем. В данной модели требуется учитывать интерфейсы компонентов, а также физические размерности входов и выходов.

Рассмотрим процесс создания системной модели. Так как разрабатываемая система содержит «физическую» (механика, электрика и т.д.) и алгоритмическую часть, то требуется учитывать их взаимосвязь. Более того, требуется учитывать соображения перехода из физических в алгоритмические части (датчики, АЦП, ЦАП). Для того, чтобы получить общее представление о системе в целом создается модель, в которой компоненты представлены в виде подсистем. При этом на данном этапе в подсистемах должен быть определен только интерфейс.

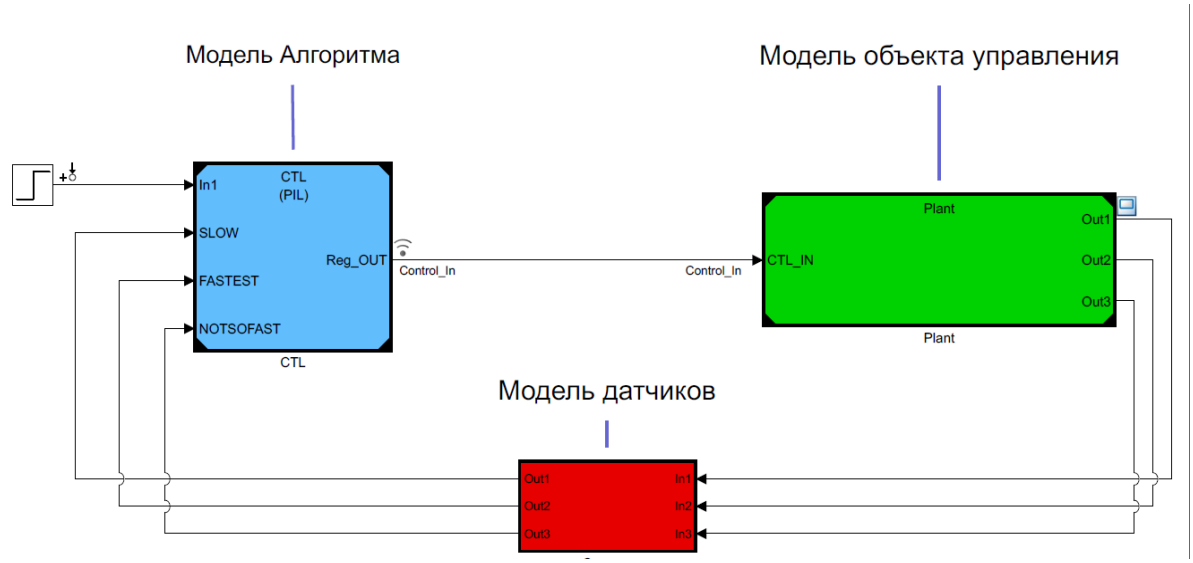

Рисунок 4 Пример системной модели

4.2 Определение интерфейса системы и компонентов

Компоненты системы и сама система обладают интерфейсами. Определение интерфейсов на ранней стадии проекта является критически важной задачей. Для того чтобы определить интерфейс компонента необходимо, но недостаточно установить следующие свойства компонента:

- Количество выходов
- Физические размерности каждого входа и выхода

Для полного определения интерфейса необходимо также определить дополнительно:

- Размерность каждого входа и выхода
- Типы данных (в случае модели алгоритма ПО)
- Диапазон значений
- Частоту дискретизации компонента

5. Процессы управления конфигурацией, трассируемостью и требованиями

Модельно-ориентированное проектирование гарантирует трассируемость между исходным кодом и требованиями высокого уровня (ТЗ)

5. 1 Управление конфигурацией

В общем случае, над системой работают несколько команд разработчиков. Это приводит к следующим “бутылочным горлышкам”:

- Несогласованность компонентов на уровне их интерфейсов.
- При передаче в системную модель компонента возможна потеря его данных (например, параметров компонентов)
- Сложность контроля изменений - затруднен анализ изменений, их влияние на систему в целом и возможность перехода к предыдущим версиям компонентов

Кроме того, отраслевые стандарты требуют применение инструмента управления конфигурацией.

В силу вышеуказанных причин требуется наличие инструмента, обеспечивающего:

- Хранение проекта с сохранением и защитой связей между компонентами, тестами, требованиями
- Привязку проекта к системам контроля версий
- Анализ проекта на наличие отсутствующих элементов

В рамках МОП, работа ведется в единой среде, и управление конфигурацией обеспечивается встроенными инструментами.

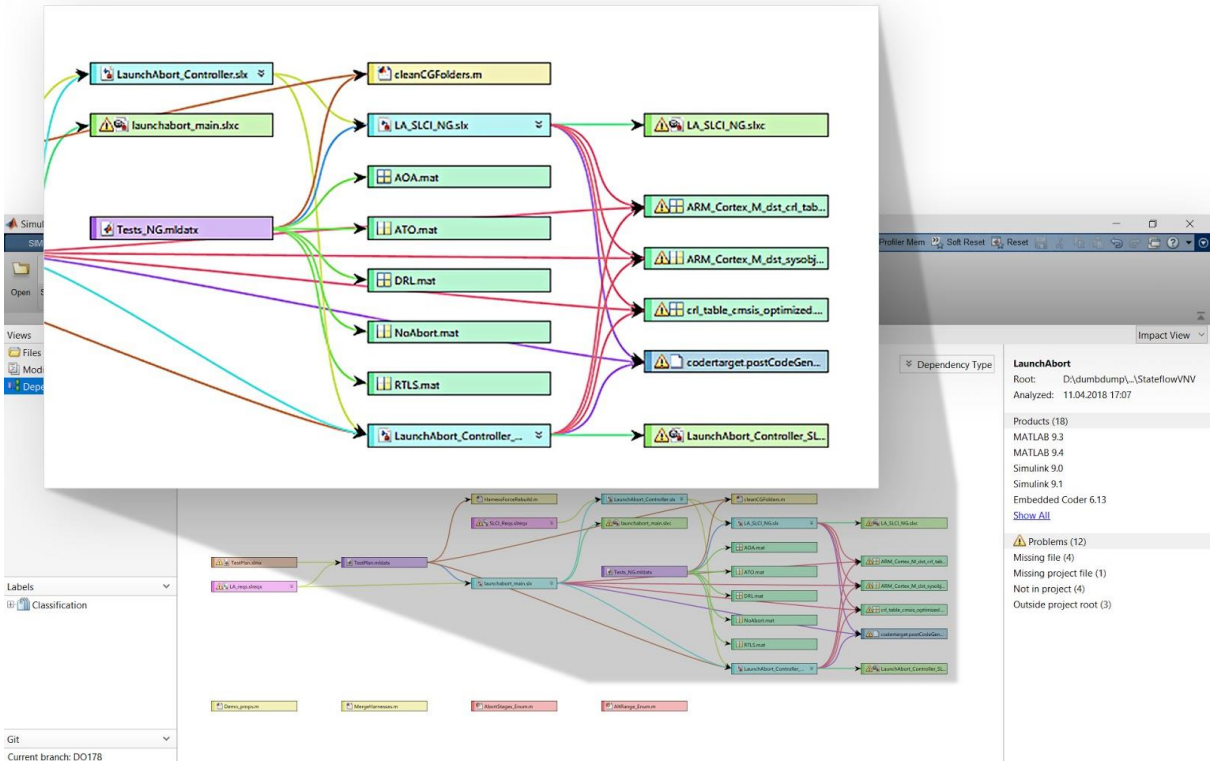

Рисунок 5 Анализ зависимостей между компонентами проекта с помощью Simulink Projects

5.2 Управление трассируемостью

5.2.1 Общие соображения

Для обеспечения целостности проекта требуется установить трассируемость между требованиями, моделями, тестовыми векторами и кодом. По мере развития проекта также выделяется отдельный процесс управления трассируемостью, задачей которого является ее поддержание. Артефактом процесса управления трассируемостью является матрица трассируемости, которая должна быть представлена в сертифицирующий орган.

5.2.2 Трассируемость «код-требования высокого уровня»

При прохождении маршрута проектирования по МОП модели привязываются к требованиям, а трассируемость между кодом и требованиям высокого уровня обеспечивается автоматически на этапе кодогенерации из модели, привязанной к требованиям.

```
106
107       case sf_counter_IN_Running:
108        /* During 'Running': '<S1>:4':
109         *  1. Нормальная работа: Нормальная работа (My#6)
110         */
111        /* '<S1>:8:1' sf_internal_predicateOutput = ... */
112        /* '<S1>:8:1' CLK == 0; */
113        if (!sf_counter_U.CLK) {
114          /* Transition: '<S1>:8' */
115          sf_counter_DW.is_c3_sf_counter = sf_counter_IN_Waiting;
116        } else {
117          /* '<S1>:9:1' sf_internal_predicateOutput = ... */
118          /* '<S1>:9:1' ACC_OUT == THRSHLD; */
119          if (sf_counter_Y.ACC_OUT == 16U) {
120            /* Transition: '<S1>:9' */
121            sf_counter_DW.is_c3_sf_counter = sf_counter_IN_Reset;
122
123            /* Entry 'Reset': '<S1>:1':
124             *  1. Состояние RESET: Состояние RESET (My#7)
125             */
```

Рисунок 7 Фрагмент сгенерированного кода, с требованиями в виде комментариев

5.2.3 Трассируемость “код-модель”

И хотя на этапе генерации исходного кода строится трассируемость «код-модель», в общем случае, стандарт предписывает подтвердить такую трассируемость независимым инструментом верификации.

70	launchabortlogic_req_full_rd_DW.is_ModeLogic =	<model>/LaunchAbortController <model>/LaunchAbortController/State ModeLogic : 153	-
71	launchabortlogic_req_fu_IN_RTLS;	<model>/LaunchAbortController/State ModeLogic : 153/State RTLS : 162	-
72		-	Nonfunctional code (Empty line)
73	/* Entry 'RTLS': '<S1>:162':	-	Nonfunctional code (Comment)
74	* 1. RTLS (Return to Launch Site) – возврат на ВПП	-	Nonfunctional code (Comment)
75	*/	-	Nonfunctional code (Comment)

Рисунок 8 Отчет о трассируемости между кодом и моделью, полученный при помощи Simulink Code Inspector

5.3 Управление требованиями

Так как модель разрабатывается в соответствии с требованиями высокого уровня и выступает как требования низкого уровня, требуется обеспечить трассируемость между этими уровнями требований. Основная сложность в обеспечении трассируемости заключается в том, что источником требований высокого уровня могут быть разнородные системы. Например, часть требований может быть создана как простой текстовый документ (например, Частное Техническое Задание), а другая часть – храниться в специализированной системе электронного документооборота (IBM Rational Doors) или системе управления жизненным циклом приложения (Polarion ALM). Очевидно, что нужен инструмент, который бы позволил агрегировать требования из разных источников в единый репозиторий, взаимодействующий непосредственно с инструментом моделирования.

Такой репозиторий будет выступать как однозначный источник требований для моделей и тестовых векторов, а жесткая интеграция требований в модели, реализованная в рамках МОП, позволяет обеспечить синхронизацию требований с их реализацией.

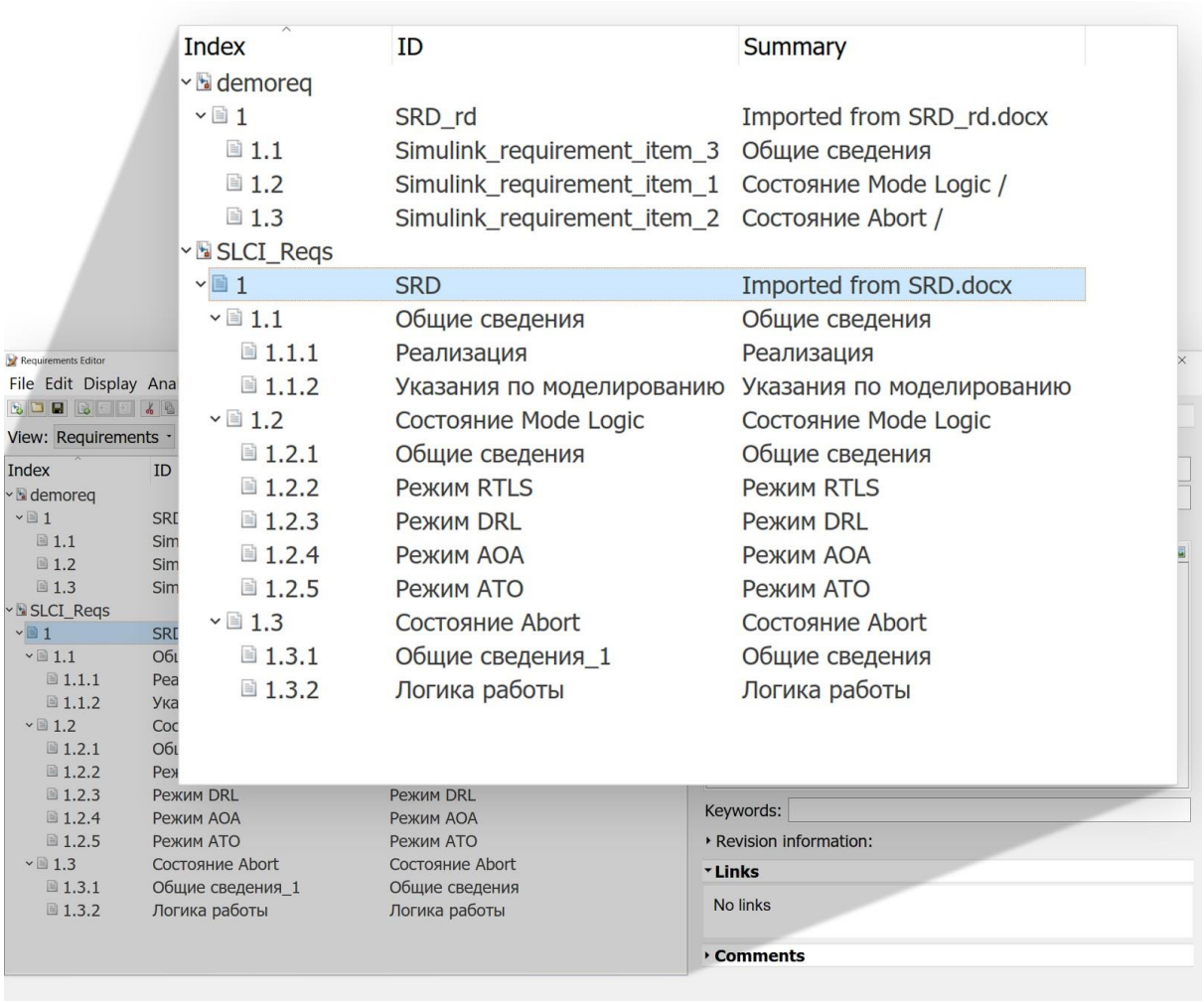

Рисунок 6 Создание и редактирование требований с помощью Simulink Requirements

6. Верификация с помощью моделей

Модельно-ориентированное проектирование автоматизирует полное тестирование алгоритма

6.1 Общие соображения

При применении МОП становится возможным организовать процесс верификации так, чтобы ошибки выявлялись на ранних этапах разработки. Для этого требуется:

- Проводить систематический анализ покрытия требованиями для обнаружения элементов модели, которые не привязаны к требованиям, и, таким образом, представляют собой посторонний функционал
- Проводить статический анализ моделей для получения их качественных характеристик и демонстрации соответствия стандартам и руководствам по моделированию
- Создать тестовые вектора, трассируемые на требования для сбора покрытия, подтверждения выполнения требований, а также для организации регрессионного анализа
- Проводить мероприятия по доказательству безошибочности модели для выявления ошибок проектирования

Так как на системном уровне в начале проектирования была проведена декомпозиция системы на компоненты, с фиксацией интерфейсов, то тестовые вектора могут быть разработаны независимо от основной модели.

6.2 Анализ покрытия модели требованиями

В рамках верификации с помощью моделей в МОП автоматически оценивается степень покрытия модели требованиями, таким образом идентифицируются потенциальные проблемы с покрытием исходного кода и выявляется посторонний функционал. Специализированное ПО может:

- Генерировать отчет о трассируемости требования-модель
- Помечать элементы модели, не трассируемые к требованиям

6.3 Статический анализ моделей

Назначение данной техники верификации модели двоякое:

- Проверка моделей на стандарты моделирования
- Построение метрик моделей

Требование к соблюдению стандартов моделирования возникает по причине того, что необходимо ограничить применение потенциально опасных конструкций и приемов моделирования, но стандарт сам по себе не гарантирует качества сгенерированного кода, так как на качество сгенерированного кода также влияет сложность модели. Поэтому, требуется обеспечить измерения и предельные значения для таких характеристик модели, как:

- Количество блоков
- Глубина вложенности
- Цикломатическая сложность

Данные характеристики показывают качество модели и являются измеряемыми в ходе статического анализа величинами.

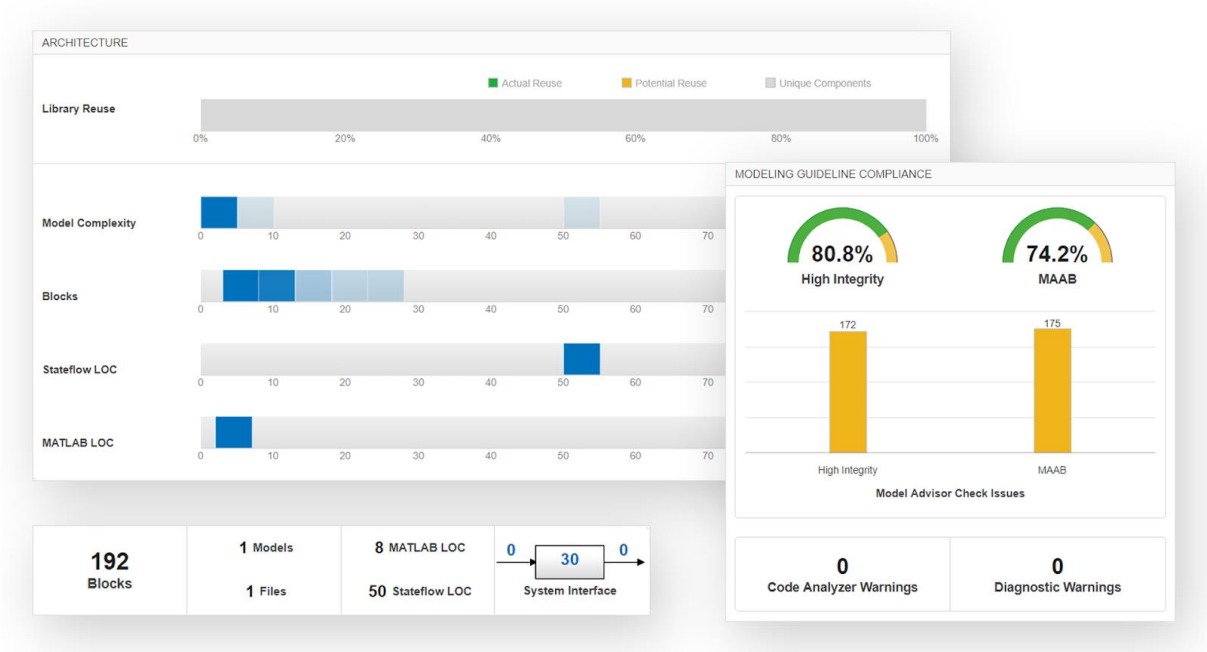

Рисунок 9 Результаты статического анализа модели выполненного в Simulink Check

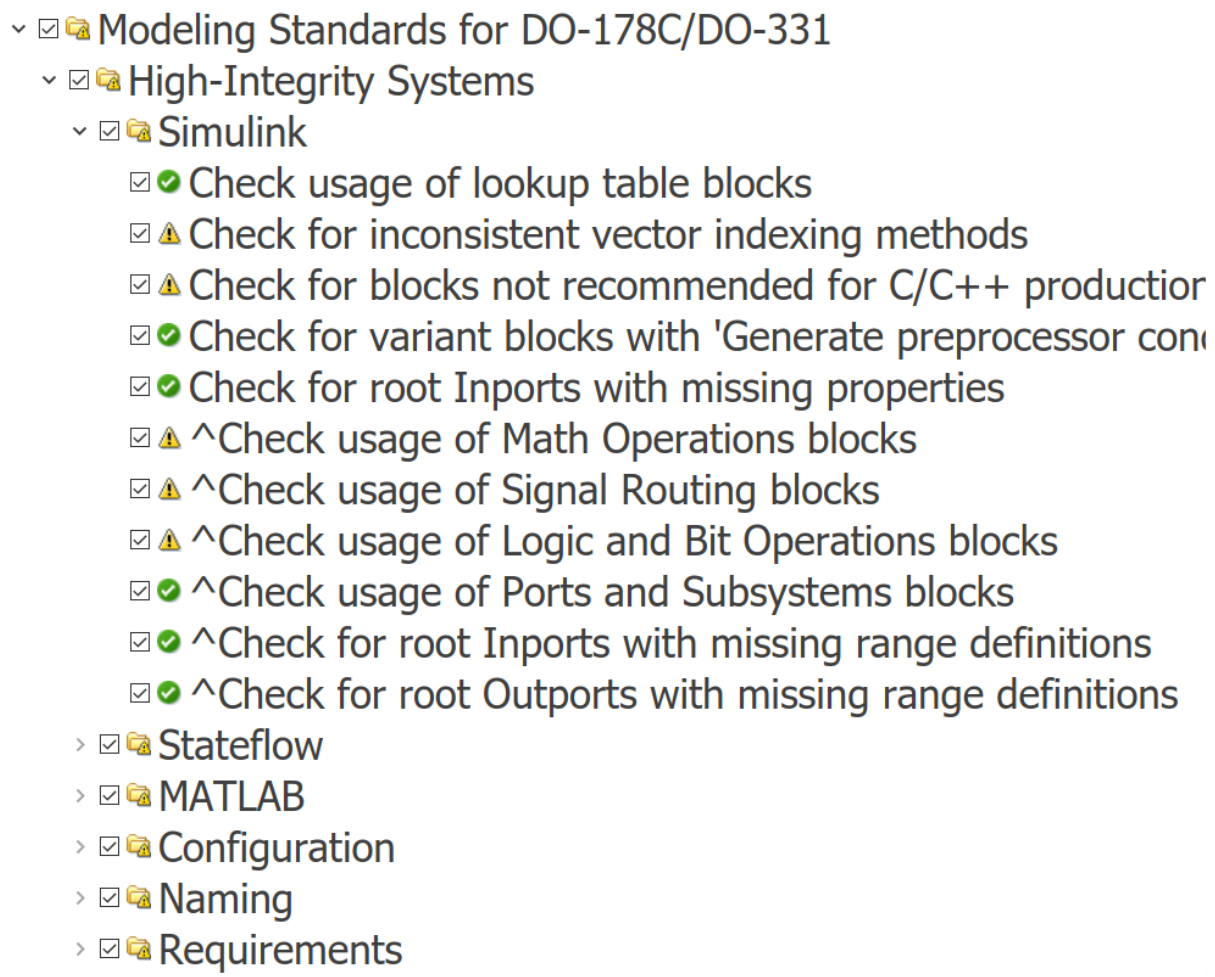

Рисунок 10 Подробные результаты статического анализа

6.4 Создание тестовых векторов и сбор покрытия тестами

Отраслевые стандарты требуют проведения тестирования и сбора покрытия ПО тестами. В контексте разработки ПО по МОП, в связи с тем, что вначале разрабатывается модель алгоритма, требования стандартов переносятся в первую очередь на модель, а интегрированные инструменты тестирования позволяют автоматизировать тестирование модели для достижения стопроцентного покрытия. Тестовые вектора для моделей должны быть трассируемы к требованиям. При создании тестовых векторов требуется также обеспечить их повторяемость. Для различных итераций моделей тестовые вектора в общем случае не изменяются, а только дополняются по мере выработки дополнительных требований.

Также требуется обеспечить тестирование на уровне как системы в целом, так и на уровне компонентов. Важно заметить, что даже достигнув полного покрытия компонента тестами, требуется провести его испытания в составе системы, так как при покомпонентном тестировании не учитываются их связи и взаимовлияние.

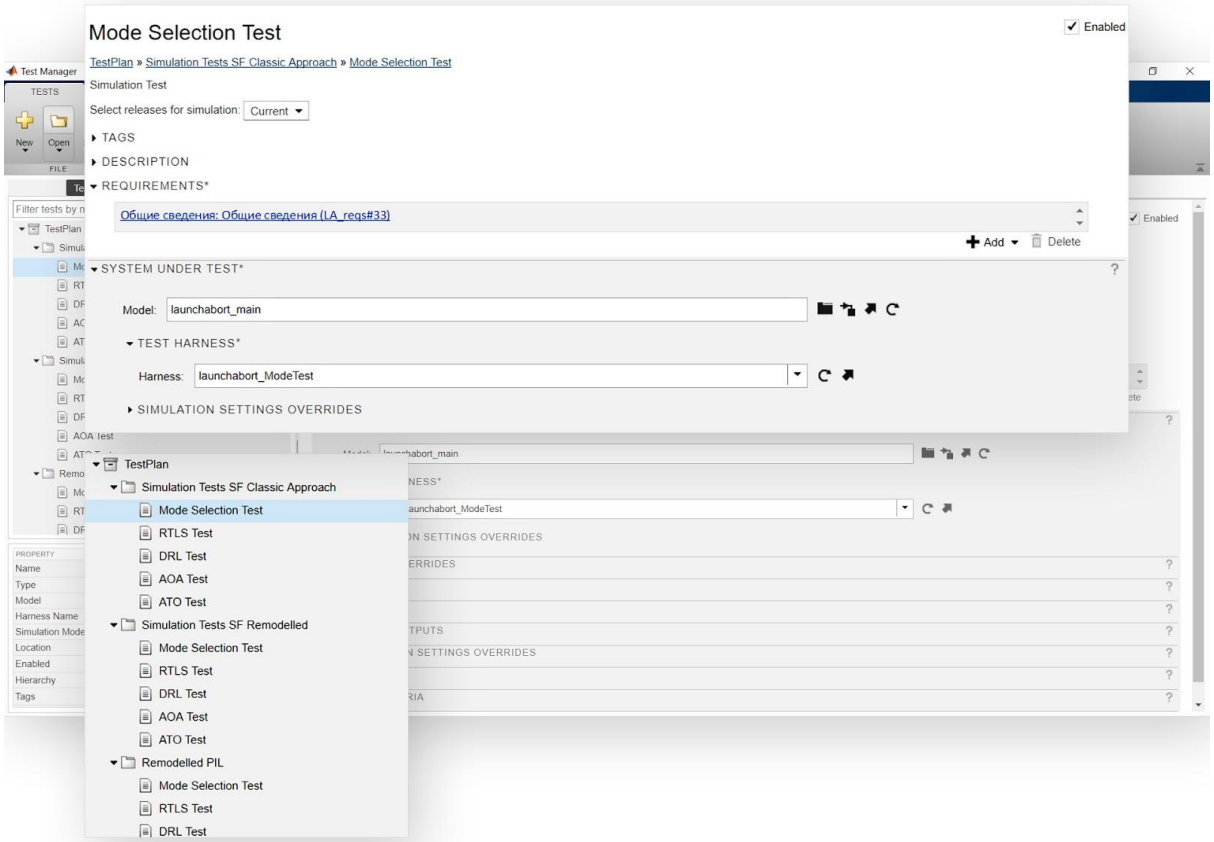

Рисунок 11 Тестовые вектора для модели, созданные в Simulink Test

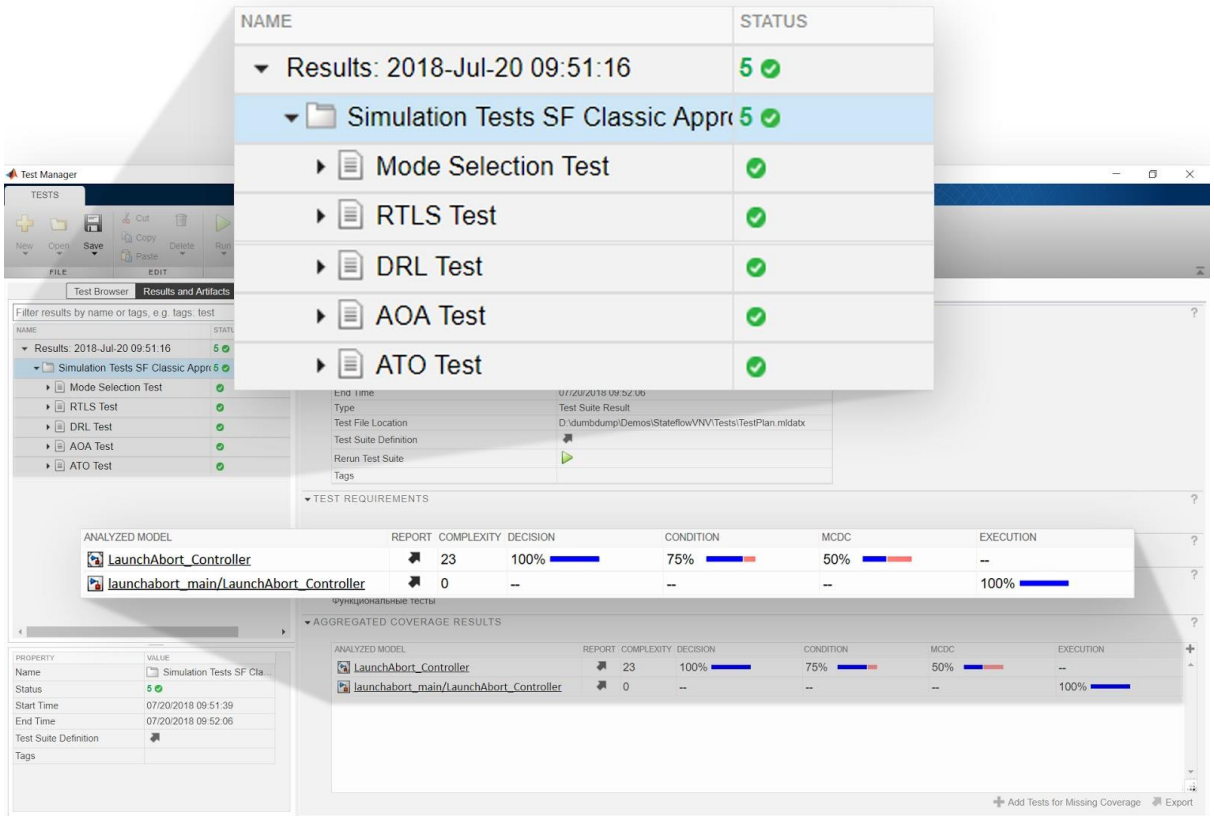

Рисунок 12 Покрытие тестами полученное в результате запуска тестовых векторов

6.5 Доказательство безошибочности модели

Функциональное тестирование со сбором покрытия является важным этапом в разработке, но такое тестирование не способно выявить плавающие неисправности и противоречивость требований.

Доказательство отсутствия плавающих неисправностей и непротиворечивости требований обеспечивается применением формальной (математической) верификации, а также доказательством выполнения требований посредством симуляции модели и моделирования требований.

Рассмотрим обоснование моделирования требований. Так как требования являются формализованными утверждениями на естественном языке, возможен «перевод» требований в математические и логические выражения, которые могут быть смоделированы. После создания таких моделей проводится формальная верификация модели компонента или системы относительно смоделированных требований для доказательства их выполнения.

Доказательство безошибочности модели проводится также на основе формальной верификации и позволяет как найти плавающие ошибки, такие как деление на ноль, так и обнаружить неисполняемые ветви модели, т.н. «мертвую логику» *(dead logic).*

Таким образом, соответствие модели становится математически строго доказанным.

7. Обеспечение качества сгенерированного кода

Современный генератор кода производит Си или HDL-код в разы быстрее человека и по скорости, потреблению ресурсов и безопасности сравнимый с рукописным

Генерация исходного кода из моделей алгоритмов является важнейшим этапом разработки. Вне зависимости от применяемого инструмента генерации кода зачастую возникают вопросы по качеству сгенерированного кода. Для оценки качества кода требуется условится о критериях качества. В настоящей статье к ним относятся:

- Использование ресурсов – потребление памяти ОЗУ и ПЗУ, быстродействие
- Функциональная безопасность
- Качество кода – соответствие стандартам кодирования, метрики кода
- Наличие ошибок времени исполнения в коде

Следовательно, требуется рассматривать все характеристики кода в комплексе и, исходя из анализа приоритетов, выбирать настройки для генератора исходного кода, что обеспечит нужный баланс между критериями.

Кроме настроек генератора исходного кода на сгенерированный код напрямую влияет сама модель. Очевидно, что чем сложнее и запутанней модель, тем сложнее и запутанней будет сгенерированный код. Поэтому в процессе разработки по МОП уделяется такое большое внимание мероприятиям верификации и валидации именно моделей (см. разд. 6).

Отлаженный рабочий процесс, включающий в себя автоматизированную верификацию моделей на всех этапах, позволяет обеспечить бесшовный переход от исходных требований к сгенерированному коду оптимального качества

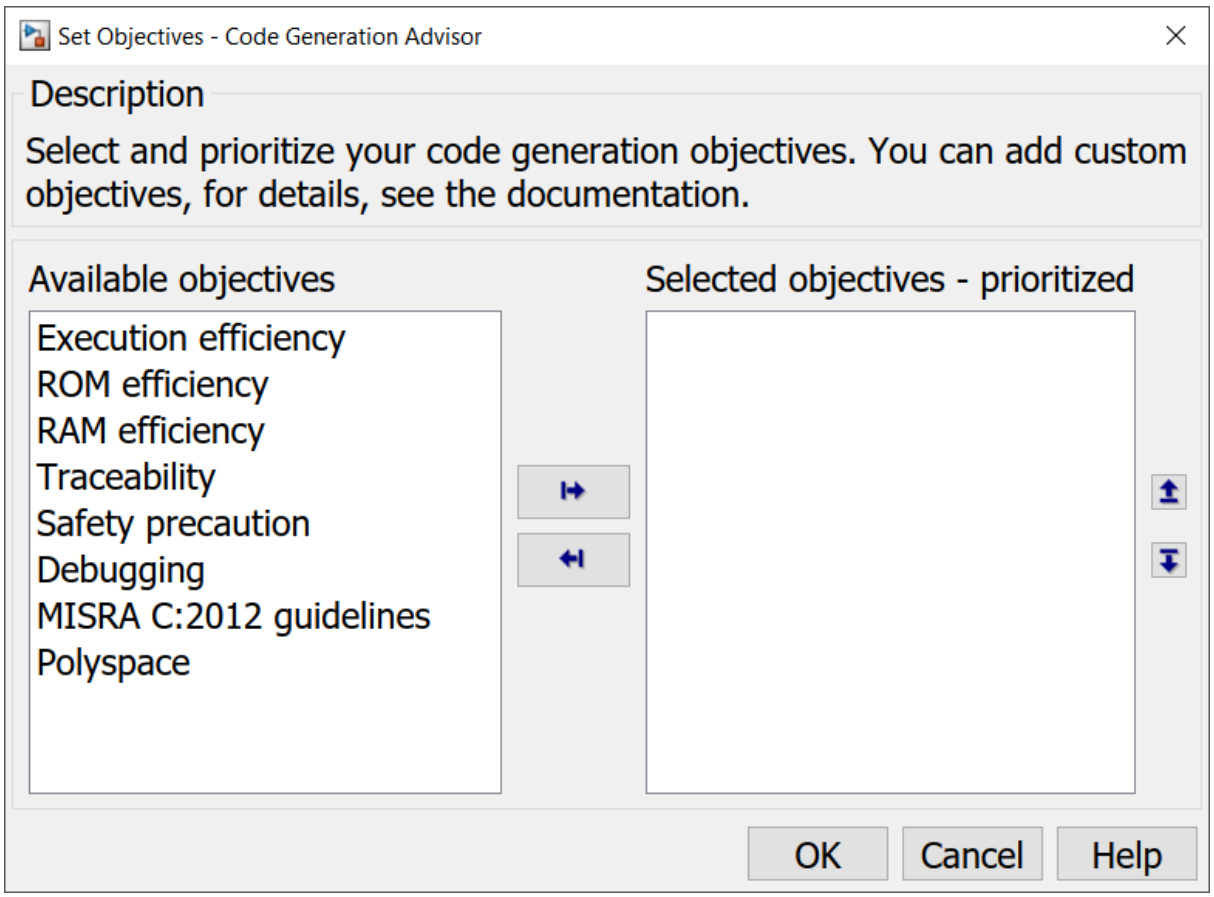

Рисунок 13 Выбор свойств генератора исходного кода

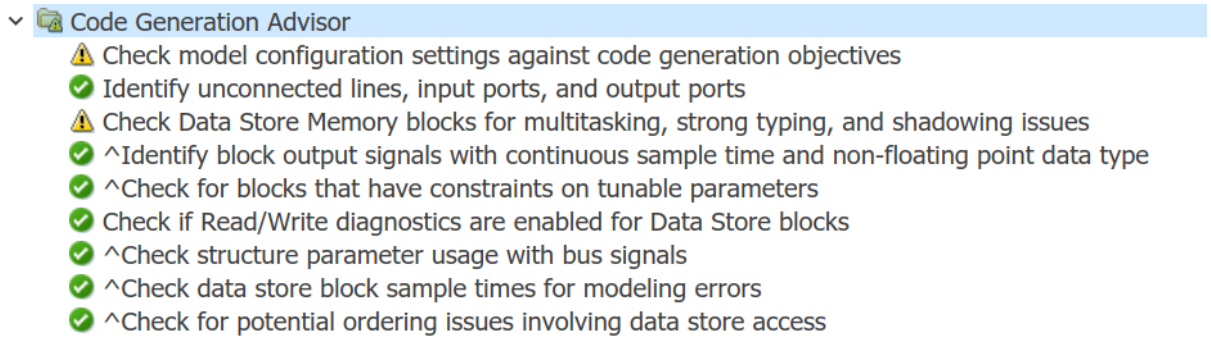

Рисунок 14 Результат проверок настроек генератора исходного кода относительно выбранных свойств

8. Верификация сгенерированного кода

<Bullit> Модельно-ориентированное проектирование автоматизирует процесс доказательства правильности кода относительно модели </Bullit>

8.1 Общие сведения

После генерации исходного кода необходимо показать следующее:

- Соответствие кода стандарту кодирования
- Эквивалентность работы модели и кода
- Трассируемость между кодом и требованиями

Вопрос трассируемости был рассмотрен ранее, поэтому не рассматривается в данном разделе.

8.2 Соответствие кода стандарту кодирования

Отраслевые стандарты требуют, чтобы исходный код был разработан однородно и был правилен и непротиворечив. Однородность кода достигается применением стандартов кодирования. Требование к непротиворечивости и правильности кода можно переформулировать как «исходный код не должен содержать ошибок времени исполнения». При использовании МОП можно с достаточной уверенностью утверждать, что если сама модель, использованная для кодогенерации не содержит ошибок времени исполнения, то и сгенерированный код тоже не будет содержать этих ошибок. Однако такое утверждение необходимо доказать. Таким образом, необходим инструмент для верификации исходного кода, который обеспечивал бы следующие функции:

- Поддержка популярных и пользовательских стандартов кодирования
- Создание отчетов о результатах работы как артефактов процесса верификации исходного кода

Концепция МОП позволяет интегрировать инструменты верификации кода со средой моделирования, что позволяет избежать ошибок, связанных с переносом артефактов между разнородными инструментами.

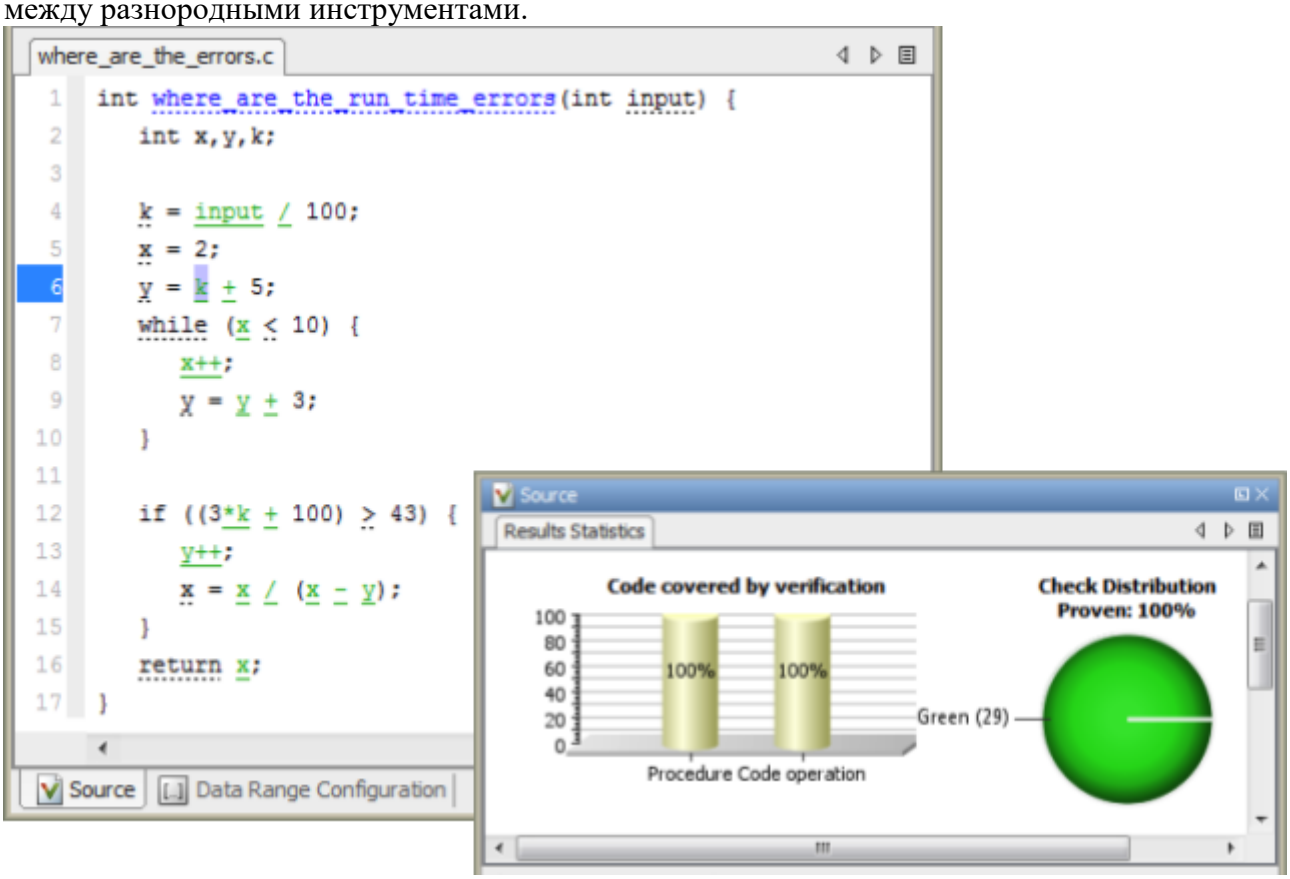

Рисунок 15 Polyspace Code Prover доказывает безошибочность кода

8.3 Доказательство эквивалентности работы модели и кода

Данный этап является критически важным в рамках МОП. Когда запускается симуляция модели, в общем случае, она работает в интерпретируемом режиме на рабочем месте разработчика (хост). Однако, исходный код, сгенерированный из такой модели будет скомпилирован и запущен на целевой платформе. Таким образом результаты работы кода и модели могут различаться из-за:

- Различий между архитектурой хоста и целевой платформы
- Оптимизаций компилятора

Поэтому требуется доказать, что результаты работы кода и модели эквивалентны. Более того, утверждение что если обеспечено 100%-ное покрытие модели тестовыми векторами,

то и сгенерированный код будет обладать той же мерой покрытия является недостаточно обоснованным в силу вышеуказанных причин.

Обоснование эквивалентности работы модели и кода осуществляется с помощью тестовых векторов, разработанных для модели и инструмента автоматизированного тестирования, рассмотренного в п.п. 6.3.
Инструмент тестирования запускает модель в режиме «Процессор-в-контуре» (PIL), агрегирует результат запуска тестовых векторов для сгенерированного кода на целевой платформе.

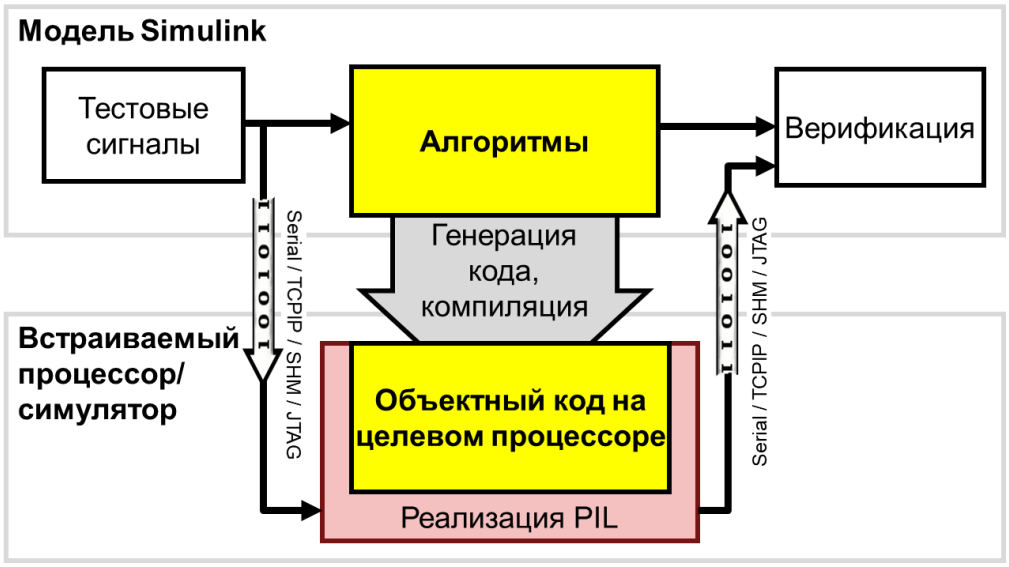

Рисунок 16 Функциональная схема метода верификации "Процессор-в-контуре"

9. Квалификация инструментов

Методология модельно-ориентированного проектирования квалифицирована в соответствии с КТ-178

9.1 Методы получения сертифицированного кода

КТ-178С предполагает наличие квалифицированных инструментов разработки, однако согласно разъяснениям [см. Р-330 "Руководство по квалификации программных инструментов"] сертифицирующих органов, **если выход неквалифицированного инструмента разработки может быть проверен при помощи квалифицированного инструмента верификации, то квалификация такого инструмента не требуется.**
Несмотря на то, является ли инструмент средством разработки или верификации, к нему применяются уровни квалификации инструмента.
Таким образом возможно получение сертифицированного кода двумя методами:

- С использованием квалифицированного генератора исходного кода
- С использованием квалифицируемого инструмента верификации

Каждый метод будет рассмотрен ниже.

9.2 Уровни квалификации

Если программные средства предназначены для автоматизации значительного числа мероприятий КТ-178С при создании доказательных артефактов, показывающих, что критерии перехода были выполнены, важно обеспечить, чтобы на эти инструменты можно было положиться. Для этого проводится квалификаци инструмента. КТ-178С заявляет, что:

«Цель процесса квалификации инструмента - гарантировать, что данный инструмент обеспечивает достоверность, по крайней мере, эквивалентную той, которая дается исключаемыми, упрощаемыми или автоматизируемыми процессами».
Квалификация инструмента является важной частью процесса сертификации, и она задокументирована в расширении P330.
P330 вводит концепцию уровня квалификации инструмента (TQL) на основе трех критериев:

1. Инструмент, результат работы которого является частью бортового программного обеспечения и, таким образом, может породить ошибку
2. Инструмент, который автоматизирует процессы верификации и, таким образом, может не обнаружить ошибку, и результат работы которого используется для обоснования устранения или сокращения процессов верификации, перечисленных в п.3
3. Процессы верификации, отличные от автоматизированных с помощью инструментов, или
    1. Процессов разработки, которые могут повлиять на бортовое программное обеспечение
    2. Инструментов, которые могут не обнаружить ошибку

Таблица 1 Уровни квалификации инструмента

Уровень ПО	Критерий		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

9.2. Сравнение методов

9.2.1. Применение квалифицированного инструмента разработки

Рассмотрим случай разработки ПО с использованием квалифицированного инструмента разработки. Это означает, что сам инструмент должен быть разработан в соответствии с КТ-178С. Возникают следующие вопросы:

- Что произойдет при обнаружении ошибки в таком инструменте?
- Можно ли говорить о том, что инструмент разработки (например, генератор исходного кода) реализующий модельно-ориентированный подход и квалифицированный по TQL-1 достаточен для получения сертификационного зачета?
- Необходимо ли проводить дополнительное тестирование ПО?
- Насколько качественным будет сгенерированный код?

Ответы на такие вопросы будут следующие:

- При обнаружении ошибки в квалифицированном инструменте разработки требуется переквалификация такого инструмента. Такая квалификация должна быть проведена совместно с разработчиком инструмента и займет значительное время
- Применение квалифицированного инструмента разработки не означает того, что исполняемый объектный код, полученный после компиляции сгенерированного

кода, автоматически верифицирован, так как сам компилятор тоже является инструментом разработки, но не квалифицированным по TQL-1
- Из ответа на вопрос 2 следует, что верифицировать исполняемый объектный код все равно необходимо, но мероприятия по верификации выходят за рамки квалификации кодогенератора
- Квалифицированные инструменты разработки должны быть достаточно простыми, так как чем больше функций, тем больше для них требований; тем сложнее квалифицировать его как инструмент разработки. Поэтому говорить с уверенностью о качестве кода невозможно

Степень выраженности описанных проблем напрямую зависит от уровня квалификации и достигает наибольшей критичности при уровне А, хотя и при прочих уровнях может стать причиной значительных задержек в разработке.

9.2.2 Использование квалифицируемого инструмента верификации

В качестве альтернативы квалифицированным инструментам разработки стандарт предлагает использовать квалифицированные инструменты верификации. Зададим аналогичные вопросы к инструментам верификации:

- Что произойдет при обнаружении ошибки в таком инструменте?
- Достаточно ли проводить квалификацию инструмента верификации для получения сертификационного зачета?
- Необходимо ли проводить дополнительное тестирование ПО?
- Насколько качественным будет сгенерированный код?

Для ответа на первый вопрос требуется рассмотреть следующие ситуации:

- Обнаружена ошибка в инструменте разработки и инструменте верификации
- Обнаружена ошибка в инструменте разработки

В первом случае, требуется обновить инструменты разработки и верификации и провести переквалификацию инструмента верификации. Такая квалификация займет меньше времени и потребует меньше взаимодействия с разработчиком инструмента.
Во втором случае требуется обновить только инструмент разработки, а повторная квалификация не требуется.

Для ответа на вопрос о достаточности квалификации инструмента верификации вновь обратимся к п. 9.1. Видно, что стандарт указывает на достаточность.

Для ответа на 3-ий вопрос достаточно заметить, что так как верификация кода - это часть процесса разработки по МОП, то может потребоваться только верификация исполняемого объектного кода на уровне ассемблеров, и только при Уровне А.

Для ответа на вопрос о качестве сгенерированного кода, следует заметить, что так как генератор исходного кода не квалифицируется, то сам генератор будет обладать широкими возможностями для оптимизации кода по требованиям разработчика.

Таким образом, видно, что применение квалифицированного инструмента разработки несет достаточные риски и не приносит ожидаемых преимуществ. Современные компании-разработчики ПО предпочитают применять подход с квалифицируемыми инструментами верификации.

10. Выводы

В статье описан подход модельно-ориентированного проектирования систем повышенной надежности. В ходе рассмотрения данного подхода были показаны его очевидные преимущества перед традиционным подходом к проектированию программного и аппаратного обеспечения:

- Применение системных моделей позволяет проверить работоспособность системы, выбрать оптимальный способ реализации и выявить ошибки технического задания на этапе проектирования
- Инструменты командной разработки гарантируют высокую степень переиспользования предыдущих наработок и упрощают взаимодействие с внешними контрагентами
- Модельно-ориентированное проектирование гарантирует соответствие проекта техническому заданию на всех этапах разработки начиная от модели и заканчивая реализацией на микропроцессоре
- Применение модельно-ориентированного проектирования гарантирует быстрый перенос алгоритмов с одной ЭКБ на другую и снимает проблему разнородности исходного кода
- Верификация и валидация являются неотъемлемой частью процесса разработки согласно МОП, что исключает необходимость отдельных мероприятий, с этим связанных
- Квалификация однородных инструментов, составляющих единую среду разработки

Таким образом, внедрение МОП будет наилучшим решением для сохранения и повышения конкурентоспособности компаний-разработчиков.

Приложение А
Список инструментов MathWorks для разработки высоконадежных систем

Этап разработки	Назначение этапа	Инструменты
Определение конфигурации ПО	Определяются методы хранения проекта, управления его версиями, состав проекта	Simulink Projects (входит в Simulink)
Разработка архитектуры модели	Система разделяется на функциональные независимые компоненты	Simulink
Разработка модели	Компоненты реализовываются, между ними устанавливается связь	Simulink
Установление трассируемости между моделью и требованиями	Устанавливается двусторонняя связь между разработанной моделью и требованиями, для доказательства отсутствия излишних конструкций и выполнения необходимых функций	Simulink Requirements
Проверка модели	Проводится статический анализ модели для документирования расхождения с руководствами по моделированию и измерения метрик модели	Simulink Check
Верификация модели	Доказывается отсутствие ошибок в модели (деление на ноль и т.д.), так же доказывается отсутствие неисполняемых элементов модели («мертвая» логика)	Simulink Design Verifier

Испытания модели	На основании требований проводится сначала покомпонентное тестирование модели, а затем и интеграционное	Simulink Test
Покрытие модели тестами	Показывается, что тестирование модели было осуществлено в полной мере, а сами тесты достаточны для доказательства корректности функционирования модели	Simulink Coverage
Получение исходного кода	Из корректной и протестированной модели генерируется исходный код на языке Си	Embedded Coder
Установление трассируемости между моделью и исходным кодом	Показывается, что код и модель соответствуют друг другу, посторонний код отсутствует	Simulink Code Inspector
Испытания кода	Тесты, созданные для модели, запускаются над кодом. Демонстрируется эквивалентность работы кода и модели.	Simulink Test
Покрытие кода тестами на целевой платформе	Тесты, созданные для модели, запускаются над кодом. Показывается полнота тестирования кода, а так же доказывается функциональная корректность кода	Simulink Coverage
Интеграция ПО	Показывается связность компонентов	Embedded Coder
Проверка исходного кода	Исходный код проверяется на соответствие стандарту кодирования, показывается отсутствие плавающих дефектов кода, а так же и неисполняемого кода (мертвого кода)	Polyspace Bug Finder, Polyspace Code Prover
Формирование отчетов	По результатам работ получаются различные документы и отчеты, содержащие артефакты процессов разработки, кодирования и верификации	Simulink Report Generator

Приложение Б
Список целей Р-331, покрываемых инструментами MathWorks

Таблица А Р-331	Критерий перехода	Пункт Р-331
MB.A-2.1	Требования высокого уровня разработаны	5.1.1.a
MB.A-2.2	Производные требования высокого уровня и предоставлены в системные процессы, включая процесс оценки безопасности системы	5.1.1.b
MB.A-2.4	Требования низкого уровня разработаны	5.2.1.a
MB.A-2.3	Архитектура ПО разработана	
MB.A-2.5	Производные требования низкого уровня и предоставлены в системные процессы, включая процесс оценки безопасности системы	5.1.1.b
MB.A-8.6	Контроль за средой жизненного цикла ПО установлен	7.1.i

MB.A-4.6	Требования низкого уровня трассируемы на требования высокого уровня	6.3.2.f
MB.A-4.2	Требования низкого уровня правильны и непротиворечивы	6.3.2.b
MB.A-4.9	Архитектура ПО непротиворечива	6.3.3.b
MB.A-4.3	Требования низкого уровня совместимы с целевым вычислителем	6.3.2.c
MB.A-4.4	Требования низкого уровня верифицируемы	6.3.2.d
MB.A-4.5	Требования низкого уровня соответствуют стандартам	6.3.2.e
MB.A-4.12	Архитектура ПО соответствует стандартам	6.3.3.e
MB.A-4.7	Алгоритмы правильны	6.3.2.g
MB.A-4.1	Требования низкого уровня соответствуют требованиям высокого уровня	6.3.2.a
MB.A-7.3	Покрытие требований низкого уровня по результатам испытаний достигнуто	6.4.4.a
MB.A-7.3	Тестовые процедуры правильны	6.4.5.b
MB.A-7.MB10	Примеры симуляции корректны	MB.6.8.3.2.a
MB.A-7.MB11	Процедуры симуляции корректны	MB.6.8.3.2.b
MB.A-7.MB12	Результаты симуляции корректны, расхождения объяснены	MB.6.8.3.2.c
MB.A-2.6	Исходный код разработан	5.3.1.a
MB.A-5.5	Исходный код трассируем к требованиям низкого уровня	MB.6.3.4.e
MB.A-5.2	Исходный код соответствует архитектуре ПО	MB.6.3.4.b
MB.A-5.3	Исходный код верифицируем	MB.6.3.4.c
MB.A-5.4	Исходный код соответствует стандартам	MB.6.3.4.d
MB.A-5.6	Исходный код правилен и непротиворечив	MB.6.3.4.f
MB.A-6.3	Исполняемый объектный код удовлетворяет требованиям низкого уровня	6.4.c
MB.A-6.4	Исполняемый объектный код является робастным по отношению к требованиям низкого уровня	6.4.d
MB.A-7.5	Тестовое покрытие структуры программы (модифицированное покрытие условий или решений) достигнуто	6.4.4.c
MB.A-6.1	Исполняемый объектный код соответствует требованиям высокого уровня	6.4.a
MB.A-6.2	Исполняемый объектный код является робастным по отношению к требованиям высокого уровня	6.4.b