УДК 004.415.532

ПОСТРОЕНИЕ ПРОЦЕССА ТЕСТИРОВАНИЯ В РАМКАХ ВЕРИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ БОРТОВОЙ АВИАЦИОННОЙ СИСТЕМЫ ВЫСОКОГО УРОВНЯ КРИТИЧНОСТИ

© 2014 О.А. Лагунков[1], К.В. Ларин[2], В.В. Шишкин[1]

[1] Ульяновский государственный технический университет
[2] ОАО «Ульяновское конструкторское бюро приборостроения»

Поступила в редакцию 22.10.2014

В статье рассматривается процесс тестирования программного обеспечения для авиационного блока. Целью является автоматизация и оптимизация процессов, окружающих непосредственное тестирование – составление тест-планов, подготовку тестовых данных и оформление артефактов процесса. Предложены модели тестируемого проекта с точки зрения верификации и методы работы с ними.

Ключевые слова: верификация, тестирование, модель проекта

Жизненный цикл программного обеспечения (ПО) высокого уровня критичности отличается от жизненного цикла обычного ПО. Одним из отличий является жестко регламентированный процесс проведения верификации. Верификация – это процесс, направленный на поиск и регистрацию ошибок. Тестирование – это один из процессов верификации, который направлен на проверку соответствия исходного или объектного кода требованиям. В данном случае под ошибкой подразумевается непредусмотренное требованиями поведение программного обеспечения.

Жизненный цикл программного обеспечения бортовых систем в авионике регламентируется документом КТ-178В, который является отечественным аналогом документа RTCA DO-178B. Данный документ подразумевает трассировку элементов каждого этапа разработки ПО – требованиями системного уровня, требованиями высокого и низкого уровней, исходным кодом и тестами.

Следует отметить, что на каждом следующем уровне разработки ПО возникают элементы, не трассируемые на элементы предыдущего этапа. Такие элементы называются производными и требуют обоснования. Например, требования низкого уровня, обоснованные методами защитного программирования.

Тестирование реализует проверку соответствия исходного или объектного кода требованиям через два механизма – покрытие требований тестами и покрытие кода тестами. Первый механизм доказывает выполнение функций, заложенных в требованиях, второй – отсутствие других функций. Документ КТ-178В, также как и DO-178B, регламентирует процесс тестирования, но не описывает методы построения и проведения процесса. В данной работе исследованы проблемы в процессах тестирования по требованиям низкого и высокого уровней и предложены методы их решения на основе разработанных моделей реализации и спецификаций.

Входными данными для проведения тестирования являются требования к ПО высокого и низкого уровней, исходный код. Трассировка между требованиями высокого, низкого уровней и исходном кодом считается обеспеченной в процессе разработки, соответственно её реализация выходит за рамки компетенций процесса тестирования и верификации в целом. Следует отметить, что требования низкого уровня должны быть декомпозированы таким образом, чтобы реализация требования укладывалась в одну функцию исходного кода.

Процесс тестирования регламентируется планами и стандартами предприятия и проекта. Для проведения тестирования предусмотрены два основных инструмента, реализующих тестовое окружение – инструмент для автоматизации тестовых прогонов на исходном коде и тестовое окружение для работы с блоком в аппаратно-программной сборке.

Лагунков Олег Александрович, аспирант
Ларин Кирилл Валентинович, главный конструктор шестого направления
Шишкин Вадим Викторинович, профессор кафедры «Измерительно-вычислительные комплексы». E-mail: shvv@ulstu.ru

Инструмент для работы с исходным кодом должен обеспечивать задачу тестовых данных – воздействий и ожидаемых реакций для конкретной функции исходного кода. Роль воздействий играют параметры, передаваемые функции и глобальные переменные, описывающие контекст выполнения. В качестве ожидаемых реакций задаются возвращаемое функцией значение (если оно есть) и глобальные переменные, которые могут измениться в ходе выполнения функции. Инструментирование кода и его выполнение должно происходить автоматически. В качестве результата теста выступает отчет об эквивалентности ожидаемых реакций и фактических, а также анализ покрытия кода.

В качестве инструмента для работы с блоком в сборке выступает стенд отладки и тестирования, который используется для задания входных данных и снятия выходных данных через аппаратные интерфейсы. Управления входами и выходами осуществляется с помощью специального языка описания тестов. При указанных исходных данных необходимо распределить тестируемые требования по тестовым окружениям, то есть отделить процесс тестирования по требованиям высокого и низкого уровней от процесса тестирования интеграции ПО и аппаратуры.

Процесс тестирования в обоих случаях требует значительных временных затрат, что также усугубляется требованиям к квалификации персонала, составляющего тесты. Таким образом, второй задачей становится построение процесса наиболее эффективным образом.

Классификация требований по типам. КТ-178В предполагает покрытие тестами всех требований высокого и низкого уровня. Пусть X – множество требований высокого уровня к программному обеспечению, Y – множество требований низкого уровня. Общий объем тестируемых требований S равен объединению всех требований: $S = X \cup Y$. Также, если разработаны и выполнены тестовый пример и соответствующая тестовая процедура для испытаний интеграции аппаратных средств и ПО или интеграции ПО, и при этом обеспечены удовлетворительное покрытие требований и структурное покрытие, то нет необходимости дублировать эти испытания испытаниями низкого уровня. Замена номинально эквивалентными испытаниями низкого уровня испытаний высокого уровня может быть менее эффективной из-за уменьшения общего объема испытываемых функций. Следовательно, требование низкого уровня при тестировании может быть заменено на трассируемое требование высокого уровня. Таким образом:

$$\forall x_n \in X \; \exists! \; Y_n \subseteq Y : Y_n \rightarrow x_n \qquad (1)$$

Отсюда следует

$$Y \rightarrow X \cup Y', \qquad (2)$$

где Y’ – множество производных требований низкого уровня. Следовательно, множество S можно представить как

$$S = X \cup (X \cup Y') = X \cup Y', \qquad (3)$$

то есть, общий объем тестирования заключается в покрытии требований высокого уровня и производных требований низкого уровня. Данный вывод не означает игнорирование требований низкого уровня, выведенных из требований высокого уровня. Составление тестов исключительно по требованиям высокого уровня невозможно, так как требования высокого уровня не привязаны к реализации. Требования низкого уровня с точки зрения тестирования показывают взаимосвязь между требуемым функционалом и исходным кодом. Таким образом, требования низкого уровня становятся справочной информацией для проецирования требования высокого уровня на исходный код.

Основным признаком для классификации требований высокого уровня является наличие зависимости выполнения требования от аппаратуры. Если требование может быть проверено только на целевой аппаратуре (например, временные задержки между изменением входов и выходов), то оно должно быть проверено в рамках тестирования интеграции ПО и аппаратуры. В других случаях предпочтительным становится использование инструмента, который обеспечивает анализ покрытия кода.

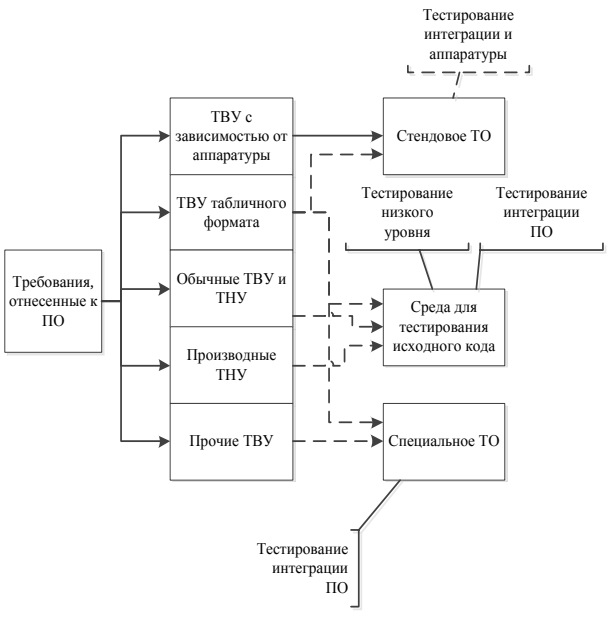

Рис. 1. Распределение требований по процедурам тестирования

Среди требований высокого уровня следует выделить требования табличного формата. Это требования высокого уровня, которые описывают логику вычислений набора выходных параметров в зависимости от значений входных параметров. Тестирование таких требований имеет потенциал к автоматизации, так как они записаны в виде выражений формальной логики с использованием конкретных вычислительных функций, описанных в исходном коде. Такие требования, как правило, переносятся в требования низкого уровня без изменений. Таким образом, получается распределение требований по процедурам тестирования, как показано на рис. 1.

К прочим требованиям высокого уровня отнесены специфичные для проектов требования, для которых более удобным может оказаться создание специального тестового окружения. На выносках отмечены виды тестирования согласно КТ-178В. В общем виде процесс тестирования в исходного кода выглядит, как показано на рис. 2.

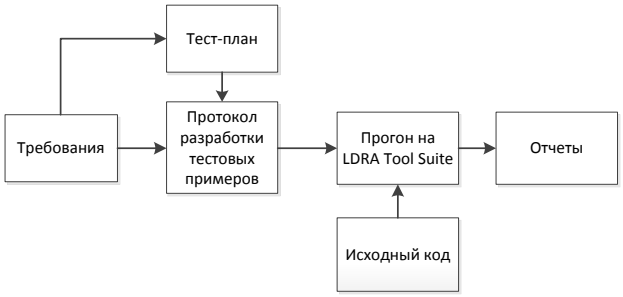

Рис. 2. Схема процесса тестирования исходного кода

Как и любая другая стадия ЖЦ ПО, тестирование по требованиям низкого и высокого уровня должно сопровождаться артефактами разработки. В данном случае артефактами являются:

1. Тест-план, содержащий в себе описание структурных элементов тестируемого объекта, их взаимосвязь и текущую информацию о ходе процесса тестирования, которая обновляется по мере выполнения.

2. Протоколы разработки тестовых примеров, которые содержат непосредственно тестовые последовательности для каждого элемента объекта тестирования. В данном случае элементом объекта тестирования является функция исходного кода.

3. Отчеты о тестовых прогонах. Отчет содержит перечни ожидаемых и фактических реакций тестируемого элемента на входы с заключением о прохождении критерия PASS/FAIL. Также в отчете содержится информация о покрытии кода по строкам и ветвлениям.

4. Анализы отчетов, в которых указана информация, необходимая разработчику для локализации ошибки или устранения мертвого кода.

На рис. 2 стрелками показаны процессы, выполняемые вручную. На основе требований составляется тест-план верификации проекта. Требования в данном случае также включают архитектуру проекта. На основе требований и тест-плана верификации составляются и заполняются тестовыми данными протоколы разработки тестовых примеров. Далее тестовые данные из протоколов переносятся в среду тестирования. Среда тестирования генерирует файл тестовой последовательности, после чего выполняется инструментирование кода, компиляция и прогон. На основе полученных отчетов составляется анализ прогона и в тест-план вносится соответствующая информация. При таком построении процесса тестирования возникает ряд проблем.

1. Составление тест-плана вручную. Для этого требуется анализ архитектурного решения проекта и построение структуры исходного кода. Также, для определения порядка тестирования функций в зависимости от выбранной стратегии (восходящей или нисходящей) необходимо анализировать исходный код и строить граф потока управления.

2. Составление протоколов разработки тестовых примеров вручную для каждой функции.

3. Перенос данных тестирования из протоколов в среду тестирования. Проблема переноса заключается в том, что в промышленном коде часто присутствуют довольно сложные конструкции, такие например, как указатель на массив указателей на структуры. Ввод таких данных в систему тестирования требует длительных манипуляций. Помимо увеличения временных затрат возрастает риск появления ошибки в тесте из-за человеческого фактора.

4. При изменении тестовых данных процедура переноса данных в систему тестирования повторяется.

5. Такой процесс при попытке распределения работы потребует дополнительные рабочие места, оснащенные средой тестирования.

Для устранения этого ряда проблем и возможности автоматизации части процесса необходимо разработать модель тестируемого объекта и инструменты для работы с ней. Тестируемый объект – исходный код проекта, разбитый на программные модули, содержащие функции (здесь и далее под функцией понимается функция исходного кода). Модель должна отражать все характеристики проекта, необходимые для создания артефактов тестирования перед тестовым прогоном. После анализа процесса составления тест-плана и протоколов разработки были выделены следующие характеристики:

1. Требования низкого и высокого уровней, а именно их идентификаторы, тексты и связи друг с другом.
2. Перечни компонент, модулей и функций исходного кода проекта и их взаимосвязь в виде иерархии.
3. Взаимосвязь функций по потоку управления.
4. Взаимосвязь функций и требований.
5. Для каждой функции – имя, тип возвращаемого значения, имена и типы передаваемых параметров, имена глобальных переменных, используемых в функции.

В качестве структурной единицы модели проекта выбрана функция, так как большинство характеристик модели связано именно с функцией, а не с требованиями. Таким образом, модель проекта представляет собой совокупность моделей функций. Модель функции обладает следующими характеристиками:

1. Имя функции.
2. Имя модуля, содержащего функцию.
3. Перечень передаваемых функции параметров и их типы.
4. Перечень глобальных переменных, которые должны быть инициализированы до выполнения функции и их типы.
5. Перечень глобальных переменных, которые могут измениться в ходе выполнения функции и их типы.
6. Тип значения, возвращаемого функцией.
7. Перечень функций, вызываемых их данной функции. Учитываются только функции, которые реализованы в данном проекте.

Для оптимизации процесса тестирования с помощью модели проекта необходимо решить ряд задач:

1. Разработать и реализовать метод генерации модели проекта на основе исходного кода и требований.
2. Разработать инструмент для работы моделью проекта и дополнить ее характеристиками тест-плана.
3. Разработать форму протокола разработки тестового примера и реализовать генерацию заготовок протоколов для функций.
4. Реализовать транслятор заполненных протоколов разработки тестовых примеров в промежуточный файл тестовой последовательности целевой системы автоматизированного тестирования.

Входными данными для процесса генерации тест-плана являются требования высокого и низкого уровней и исходный код проекта. Выделенные ранее характеристики функции можно получить, анализируя исходный код проекта. Выходными данными анализа должен стать список функций, отнесенные к ним идентификаторы требований и граф потока управления. После интеграции результата анализа потока данных в заготовку, модель наполняется требованиями.

В качестве платформы для реализации выбран .NET Framework и язык Visual C#, что обусловлено простой интеграцией с офисными приложениями. В качестве метода хранения модели выбран файл разметки XML. Выбранная платформа обладает встроенными методами сериализации и десериализации экземпляров классов в xml структуры. Разработан синтаксический анализатор, который получает на вход исходный код проекта, составляет перечень функций, описанных в проекте, и строит граф вызова функций друг другом. Пример простейшего графа показан на рис. 3.

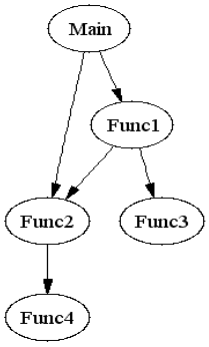

Рис. 3. Пример графа потока управления

Также синтаксический анализатор включает в модель ссылки на требования в виде идентификаторов, основываясь на информации о трассировании исходного кода. Модель сериализуется в набор экземпляров класса, каждый из которых представляет функцию со всей информацией, характеризующую проект с точки зрения спецификации. Для работы по заполнению тест-плана создана графическая оболочка, позволяющая редактировать поля XML структуры, добавляя в неё информацию об исполнителе, дате тестирования, результатах и ссылками на артефакты тестирования.

Протокол разработки теста представляет собой файл Microsoft Excel, состоящий из нескольких листов. На первом листе указывается информация о составителе теста, файлах исходного кода, на которых выполняется прогон, а также журнал изменения протокола. Второй лист содержит тестовые примеры, структура представлена на рис. 4.

Третья страница шаблона содержит идентификаторы и тексты требований, реализованных в функции. После заполнения шаблона тестовыми данными с помощью разработанного плагина для MS Excel происходит трансляция протокола

в промежуточный файл тестовой последовательности для целевой системы автоматизированного тестирования. Полученный файл готов к тестовому прогону на исходном коде. После анализа отчетов тест-план заполняется соответствующей информацией, на этом итерация тестирования заканчивается.

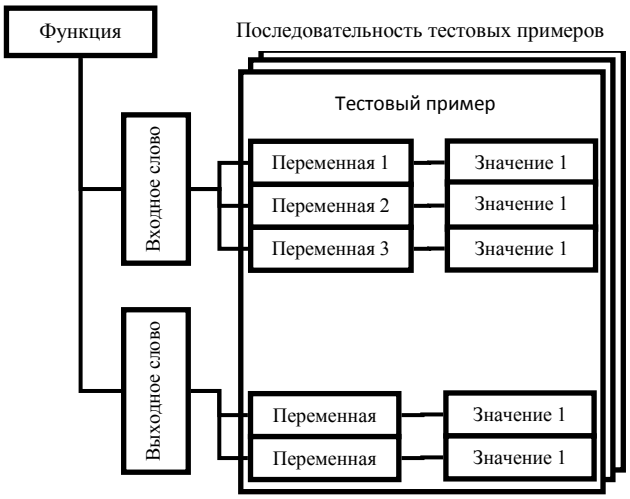

Рис. 4. Структура тестового примера.

Ранее отдельно были упомянуты требования высокого уровня табличного формата. В зависимости от предназначения тестируемого блока их количество сильно отличается. Под требованиями табличного формата подразумевается таблица или совокупность таблиц, описывающих реакции блока на воздействия. Воздействия и реакции передаются и снимаются по внешним интерфейсам. Требование описывает зависимость между входами и выходом в виде функции: $F(X) = y$, где X – это множество входных параметров, а y – выход, зависимый от указанных входов. Функция записывается с помощью логико-математического выражения с использованием операторов, определенных в требованиях. В одном блоке может быть задействовано больше тысячи входных и выходных интерфейсов со сложными функциональными зависимостями. Время написания тестов по таким требованиям существенно возрастает при использовании обычной методики подготовки входных и ожидаемых выходных данных. Это обусловлено сложностью вычисления выходных данных.

Для решения этой проблемы можно применить подход, не требующий задания ожидаемых выходных данных. Для этого необходимо создать модель тестируемого объекта и сравнивать её реакции с реакциями объекта. Может показаться, что для этого потребуется полностью повторить написание исходного кода блока и доказать правильность его выполнения, но это не так. Тестируемый объект, как правило, обладает множеством других функций, помимо того, что описано в табличных требованиях. Также при создании модели не накладываются ограничения на аппаратную платформу и реализацию. Например, сама по себе реализация операторов вычисления на языке программирования высокого уровня или на функциональном языке будет достаточно тривиальна, таким образом, рассмотрения исходного кода модели кажется достаточным для ее принятия. Схематичное представление разработанной модели представлено на рис. 5.

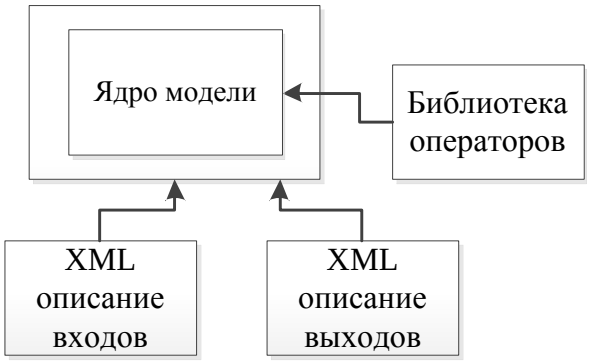

Рис. 5. Модель тестируемого блока.

XML описания входов и выходов содержат всю информацию, необходимую для распаковки входных и упаковки выходных данных в форматы интерфейсов блока. Также в XML описании выходов хранятся непосредственно выражения функциональной зависимости от входов. Библиотека операторов содержит реализацию операторов вычисления проекта на языке программирования. Ядро модели представляет собой своего рода компилятор функциональных выражений. Ядро окружает интерфейсная оболочка, отвечающая за упаковку и распаковку данных, которая взаимодействует с XML описанием параметров и передает в ядро унифицированную информацию. При смене тестируемого блока все компоненты модели кроме ядра могут быть изменены в зависимости от специфики блока.

В качестве инструмента взаимодействия с моделью выступает графическая оболочка, позволяющая в удобной форме задавать входные данные для модели. Также разработан компонент, отвечающий за генерацию кода для ПДК «Фрегат» на основе заданных для модели входов и полученных выходов. При дальнейшем развитии модели возможно также увеличение эффективности за счет оптимального распределения тестируемых параметров в тестовых файлах и параллельного прогона нескольких тестов независимых входов.

После анализа регламентирующих документов и специфики тестовых окружений выявлены критерии по распределению требований по

средам тестирования. Анализ работы пакета автоматизированного тестирования выявил ряд проблем, приводящих к уменьшению эффективности процесса. Разработана модель тестируемого проекта. Работа с моделью позволяет уменьшить время, затрачиваемое на выполнение процедур, не связанных с составлением тестовых последовательностей. Выделен отдельный тип требований – табличные требования, описывающие функциональные зависимости внешних интерфейсов тестируемых блоков. Составление тестов по таким требованиям, а именно вычисление ожидаемых реакций представляет сложность для исполнителя. Разработана модель, которая позволяет получать эталонные реакции тестируемого блока на заданные входные данные. Именно эти реакции играют роль ожидаемых выходных данных. Разработанные инструменты для взаимодействия с моделями позволяют исполнителю абстрагироваться от конкретной реализации тестового окружения и архитектуры тестируемого блока. Такая абстракция позволяет составлять тестовые последовательности несколькими исполнителями одновременно без привязки к конкретным рабочим местам на производстве.

СПИСОК ЛИТЕРАТУРЫ:

1. *Kondratiev S.A., Lagunkov O.A., Shishkin V.V.* Automatic Test Cases Generation Method for Embedded Systems Software / *S.A. Kondratiev, O.A. Lagunkov, V.V. Shishkin* // International Congress on Information Technologies-2012 (ICIT-2012): Proceedings of Abstracts. – Saratov : SSTU, 2012.
2. КТ-178. Квалификационные требования. Требования к программному обеспечению бортовой аппаратуры и систем сертификации авиационной техники. — Межгосударственный авиационный комитет, Авиационный регистр, 1996.
3. LDRA Tool Suite, 2014. Retrieved July 14, 2014, URL: http://www.ldra.com/en/software-quality-test-tools/group/by-product-module/ldra-tool-suite
4. RTCA/DO-178B. Software considerations in airborne system and equipment certification -RTCAInc, 1992.
5. *Шишкин, В.В.* Автоматизированная система создания диагностического обеспечения комплексных систем электронной индикации и сигнализации летательных аппаратов / *В.В. Шишкин, С.В. Черкашин* // Датчики и системы. 2007. №12 (103). С. 39-42.
6. *Шишкин В.В., Черкашин С.В.* Автоматизация проектирования диагностического обеспечения и диагностирования авиационных бортовых информационных систем / *В.В. Шишкин, С.В. Черкашин.* – Ульяновск: УлГТУ, 2010. 210 с.

ORGANIZATION OF TESTING IN VERIFICATION OF AVIONICS SOFTWARE IN ONBOARD SYSTEMS WITH HIGH CRITICAL LEVEL

© 2014 O.A. Lagunkov[1], K.V. Larin[2], V.V. Shishkin[1]

[1] Ulyanovsk State Technical University
[2]Ulyanovsk Instrument Manufacturing Design Bureau

In this article we consider problem of organization of testing based high- and low-level requirements. Purpose is to make processes that surround testing more effective such as test-plan composing, preparing test data and collecting artifacts. New models of testing project in terms of verification process and methods for working with models are proposed.

Key words: *verification, testing, requirements, project model*

Oleg Lagunkov, Post-graduate Student
Kirill Larin, Chief Designer of Sixth Department
Vadim Shishkin, Professor at the Department "Measuring and Computing Systems". E-mail: shvv@ulstu.ru