УДК 681.518.5

Г.С. РАНЧЕНКО, Д.И. ВОЛКОВ, В.В. ДАНИЛОВ

[1]АО «Элемент», Одесса

ПРОГРАММНЫЙ СИМУЛЯТОР ДЛЯ РАЗРАБОТКИ И ПРОВЕРКИ АЛГОРИТМОВ ФУНКЦИОНИРОВАНИЯ САУ

Представлен цикл тестирования и отладки алгоритмов управления и диагностирования цифрового регулятора ГТД типа FADEC. Рассмотрены практические аспекты построения и последующего использования программного симулятора совместной работы САУ и ГТД. Обоснована необходимость применения программного симулятора работы двигательных установок под управлением цифровых регуляторов, в том числе для обеспечения требований КТ-178. Обозначены роль и место программного симулятора в отладке и тестировании ПО цифровых регуляторов. Рассмотрены отдельные вопросы модульного и компонентного тестирования. Статья может быть полезна программистам, разрабатывающим ПО для FADEC, тестировщикам, а также специалистам из смежных областей.

Ключевые слова: *FADEC, САУ ГТД, цифровой регулятор, программное обеспечение, программный симулятор, непрерывная интеграция, тестирование, устойчивость САУ.*

Введение

Программное обеспечение современных САУ типа FADEC является составляющей, на которую приходится основная часть трудозатрат, а также ответственности за отказы или их парирование. Кроме того, характерной особенностью является то, что трудозатраты имеют место в течение всего жизненного цикла САУ, равно как и то, что дефекты, приводящие к отказам, могут вноситься в течение всего жизненного цикла САУ. Соответственно, правильная организация процесса разработки ПО и применение надлежащего инструментария является необходимым условием, как для повышения экономической эффективности, так и для повышения надежности изделия.

Важным является уровень автоматизации испытаний, время итерации при выпуске новой версии ПО, что определяет экономическую эффективаность процессов разработки и сопровождения ПО цифровых регуляторов.

В целом в статье рассматриваются общие вопросы, но в некоторых случаях приводятся конкретные примеры их использования в АО «Элемент», в том числе для РДЦ-450М – цифрового регулятора двигателя двухдвигательной вертолетной установки на базе ГТД АИ-450М.

1. Постановка задачи

Тестирование и отладка алгоритмов начинается на этапе получения требований или их изменения. Для алгоритмов управления, диагностирования, цифровой фильтрации и т.д. применяются пакеты математического моделирования (рис. 1) АО «Элемент» главным образом использует Matlab & Simulink и LabView. Логические алгоритмы анализируют с использованием абстракций блок-схем, машин состояний, диаграм последовательностей, с финальным составлением таблиц состояний и переходов. В процессе разработки или модификации ПО выполняется его тестирование, а также программная симуляция работы двигательных установок под управлением цифровых регуляторов. Далее выполняется уже программно-аппаратная симуляция работы двигательных установок под управлением цифровых регуляторов с использованием стенда-имитатора. Следующий этап – стендовые и летные испытания [1].

Одним из распространенных заблуждений является то, что на этапе, когда цифровой регулятор разработан, пройдены стендовые и тем более летные испытания, нет необходимости в чисто программной симуляции работы двигательных установок под управлением цифровых регуляторов. Иными словами чисто программная симуляция воспринимается, как вынужденная мера на этапе отсутствия аппаратных средств и объекта управления, необходимость которых отпадает сразу же после появления последних.

Задачей данной статьи является опровержение данного заблуждения, разъяснение роли и места программной симуляции САУ и ГТД. Кроме того, уточняется отличие решаемых задач от модульного и компонентного тестирования, в частности, специфических, таких как проверка устойчивости контуров управления.

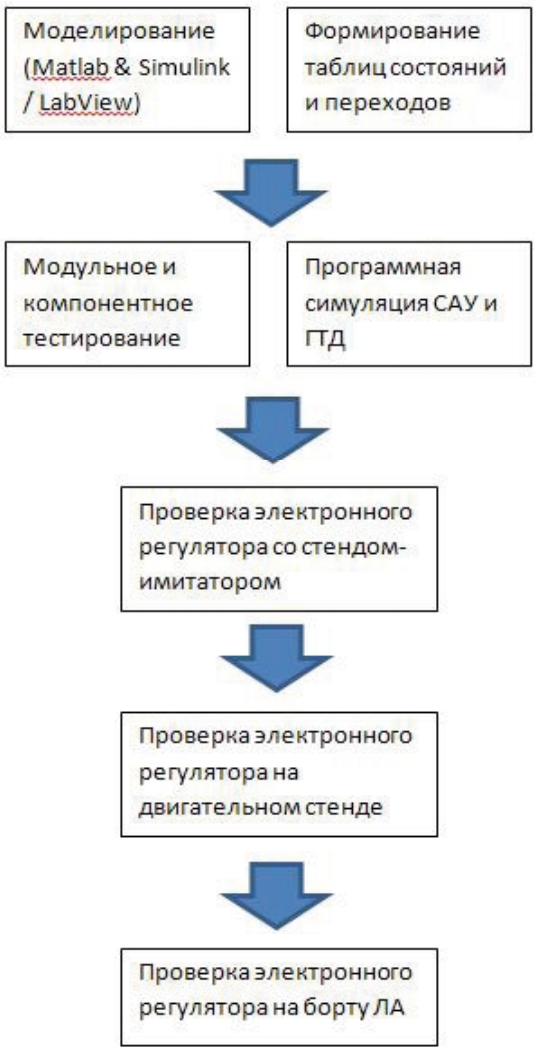

Рис. 1. Тестирование и отладка алгоритмов

2. Программный симулятор

Первым делом постараемся обозначить, что понимается под программным симулятором и чем он отличается от модульных (unit) или компонентных тестов.

Итак, модульные тесты это тесты, которые обеспечивают тестирование некоторого модуля. Разные разработчики, в том числе в зависимости от используемого языка программирования, понимают под модулем метод (функцию) при функциональном программировании или иногда отдельный файл, который агрегирует эти методы, либо класс, при использовании объектно-ориентированного программирования (ООП). Опять-таки, некоторые разработчики не выделяют отдельно компонентные тесты, считая это вариантом модульных тестов. Однако желательно их отличать, так как от этого зависит подход к тестированию, требования к тестированию и т.д. Считаем, что модульное тестирование обеспечивает тестирование некоторой сущности, будь то класс, метод или группа методов, как белого ящика, с обязательным контролем внутренних (приватных) и внешних данных, в том числе объема используемой памяти, и возвращаемых результатов. При этом внешние интерфейсы, если и задействуются, то лишь с целью протестировать их непосредственную работоспособность, т.е. ввод-вывод данных и выполнение вызовов. При компонентном же тестировании объектом тестирования является компонент – сущность, для которой специфицированы требования и внешние программные интерфейсы. Тестирование компонента выполняется как для «черного ящика», т.е. внутреннее состояние компонента не анализируется. Такое разделение позволяет, в первом случае выполнить сугубо программные вещи и нефункциональные требования, фокусируясь на надежности, использовании памяти, быстродействии, а во втором – формализовать проверку компонента на соответствие требованиям, не загромождая тестирование нюансами внутренней организации компонента. Кроме того, при анализе степени покрытия тестами, в случае модульного тестирования главным показателем является процент кода, который затрагивается в процессе выполнения всего набора тестов, в то время, как при выполнении компонентных тестов этот показатель используется лишь как вспомогательный. Покрытие кода проверяется, например, с помощью gcov, достаточно удобного инструмента. Из замеченных неудобств лишь то, что при распределении тестов для одного модуля на несколько исполняемых файлов, результаты также представлены несколькими файлами. Хотя, впрочем, объединение результатов, делается при помощи примитивного скрипта. Непокрытый код – либо недоработка при написании тестов либо неиспользуемый код, подлежащий удалению. Для компонентного же тестирования, наоборот, показателями являются степень покрытия тестами интерфейсов (API) компонента, а также покрытие функциональных требований. Хотя впрочем, последний показатель может быть использован и при модульном тестировании в случае, когда требования относятся к алгоритмам, численным методам и пр.

Что же такое программная симуляция и, чем она отличается, например, от компонентного тестирования? В каком-то смысле это действительно нечто вроде большого компонентного теста, в который вовлекается значительная часть рабочего ПО САУ, но, во-первых, это уже не отдельный компонент, а их совокупная работа, во-вторых, имеется свой специфический

для САУ набор выполняемых проверок, причем с использованием математических моделей [1] двигателей, агрегатов, редуктора и винта в случае вертолетной ДУ. Проверяются показатели качества регулирования или ограничения параметров, устойчивость контуров в части задач управления, работа инженерных алгоритмов и алгоритмов диагностирования.

В принципе набор выполняемых проверок практически совпадает с таковым при проверке цифровых регуляторов с использованием стенда-симулятора, но:

– время применения обновленного кода 10 секунд против минимум десятков минут при прошивке цифрового регуляторов,

– применение обновленного кода автоматизируется, что является необходимым условием для организации непрерывной интеграции (continuous integration),

– тестирование можно выполнять в убыстренном масштабе времени, что при использовании современных вычислительных мощностей означает быстрее минимум на 2 порядка, даже не учитывая возможности распараллеливания,

– тестирование можно выполнять параллельно на нескольких компьютерах, а также использовать преимущества многоядерных процессоров, запуская тесты в несколько потоков,

– можно динамически модифицировать код и в том числе можно выполнять проверку устойчивости контуров управления на разомкнутом контуре, что, например, актуально для проверки по критерию Найквиста-Михайлова,

– можно проще интегрировать внешние специализированные программные средства, например, Matlab & Simulink для проверки устойчивости,

– отсутствуют ограничения по логированию отладочной информации,

– можно выполнять все вышеперечисленные операции удаленно при помощи VPN, что актуально, например, при работе у Заказчика.

Ну и железобетонный аргумент – программный симулятор все равно приходится разрабатывать на этапе, когда нет ни аппаратных средств, ни ГТД для проведения испытаний и т.д. Почему же не сделать хорошо и не получить вышеперечисленные возможности?

Непрерывная интеграция это – современная практика разработки ПО, без которой практически не обходится ни один разработчик ПО высокой сложности, такого как применяемого в т.ч. FADEC. Другой вопрос, насколько автоматизирован процесс. Даже, если просто предусмотрена еженедельная интеграция, инициируемая в ручном режиме, все равно это концепция непрерывной интеграции и практически наверняка процесс сборки и тестирования в значительной мере автоматизированы. В противном случае тысячи проверок, что вполне характерно для любых систем уровня FADEC по сложности, не говоря уже об ответственности системы, будут выполняться месяцами.

Непрерывная интеграция в целом характеризуется рис. 2.

Рис. 2. Непрерывная интеграция

В зависимости от применяемого подхода по расписанию или по внесению изменения, ПО соответственно с накопленными в системе контроля версий с изменениями или с единственным изменением, являющегося триггером для запуска цикла, собирается на build сервере затем запускаются тесты, в том числе с использованием программного симулятора и затем публикуется новая рабочая версия (все подписчики уведомляются о выпуске новой версии и результатах тестирования).

В случае, если сборка или тестирование неуспешны, и, в зависимости от настроек, процесс прерывается и производится уведомление по электронной почте ответственных лиц, в т.ч. интеграторов проекта, а также лица (лиц), вносивших изменения после предыдущей успешной сборки или тестирования.

Кроме непосредственно тестирования ПО в классическом смысле выполняется проверка устойчивости контуров управления и ограничения. Устойчивости контуров проверяются на всех режимах, с различными положениями механизации, а в случае двухдвигательной силовой установки, как в случае АИ-450М, также с различными комбинациями работы двигателей в т.ч. при работе двигателя под управлением гидромеханического резерва:

– один работающий двигатель под управлением цифрового регулятора, второй выключен;

– два работающих двигателя под управлением цифровых регуляторов;

– один работающий двигатель под управлением цифрового регулятора и второй работающий двигатель под управлением гидромеханической резервной САУ.

Учитывая, что программный симулятор позволяет проведение активного эксперимента, причем в отсутствии шумов, как в случае использования стенда-имитатора, то построение, например, Bode диаграммы, на основании быстрого преобразования Фурье fft отклика на единичный импульс (дельта-функцию) вполне тривиальная задача при интеграции симулятора с Matlab. Причем, учитывая, что задачей является не столько полноценный анализ, который выполняется в Matlab на этапе дизайна алгоритма, сколько допусковый контроль на соответствие результатам полного анализа по реперным точкам и последующий визуальный контроль сформированных отчетов. Грубо говоря, цель проверить, что ничего не «поломали».

Программный симулятор на примере регулятора РДЦ-450М и ГТД АИ-450М состоит из следующих частей:

– немодифицированное рабочее ПО блоков РДЦ-450М, два двухканальных электронных блока, т.е. четыре компонента с рабочим ПО;
– два компонента модели ГТД АИ-450М;
– компонент модели редуктора и вертолетного винта;
– менеджер симулятора, который обеспечивает информационный обмен и синхронизацию выполнения отдельных частей ПО;
– ПО КПА-450М, обеспечивающие визуализацию и регистрацию параметров работы моделей и цифровых регуляторов.

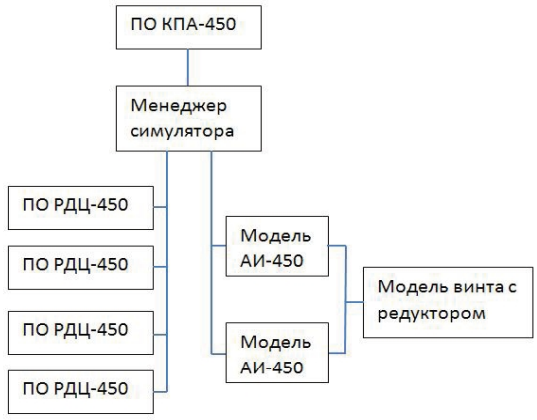

Рис. 3. Структура программного симулятора работы двухдвигательной вертолетной силовой установки на базе ГТД АИ-450 под управлением РДЦ-450.

Заключение

Итак, расмотрены основные стадии проверок ПО, используемого в САУ типа FADEC, рассмотрены причины использования программного симулятора совместной работы САУ и ГТД с использованием оригинального кода. Показана его уникальная роль, не пересекающаяся с классическими модульным и компонентным тестированием, в том числе в процессе непрерывной интеграции. Также вкратце рассмотрена структура программного симулятора, показывающая простоту решения, контрастирующую с массой извлекаемых преимуществ и новых возможностей.

Литература

1. Стенды-имитаторы и их применение на различных стадиях жизненного цикла систем управления газотурбинных двигателей [Текст] / Д. И. Волков, В. М. Грудинкин, В. А. Качура [и др.] // Авиационно-космическая техника и технология. – 2008. – №9 (56). – С. 133–137.

2. Миргород, В. Ф. Новые компьютерно-ориентированные формы математических моделей авиационных газотурбинных двигателей [Текст] / В.Ф. Миргород, В. М. Грудинкин // Искусственный интеллект. – 2008. – №4. – С. 117–124.

Поступила в редакцию 17.05.2016

Г.С. Ранченко, Д.І. Волков, В.В. Данілов. Програмний симулятор для розробки та перевірки алгоритмів функціонування САУ

Представлено цикл тестування та налагодження алгоритмів управління та діагностування цифрового регулятора ГТД типу FADEC. Розглянуто практичні аспекти побудови та наступного використання програмного симулятора сумісної роботи САУ та ГТД. Обґрунтовано необхідність використання програмного симулятора роботи двигунових установок під управлінням цифрових регуляторів,також задля забезпечення вимог КТ-178. Визначено роль та місце програмного симулятора у налагодженні та тестуванні ПЗ цифрових регуляторів. Розглянуто окремі питання модульного та компонентного тестування. Стаття може бути корисною програмістам,що розробляють ПЗ для FADEC, тестувальникам,а також фахівцям з суміжних областей.

Ключові слова: *FADEC, САУ ГТД, цифровий регулятор, програмне забезпечення, програмний симулятор, безперервна інтеграція, тестування, стійкість САУ, АИ-450М.*

G.S. Ranchenko, D.I. Volkov, V.V. Danilov. Software simulator for fadec algorithms development and testing

The article describes the testing and debugging process for FADEC. There are practical aspects of software simulator development and usage. The article proves that it is necessary to use software simulator in order to meet DO-178 requirements. Unit and component testing is described. The article may be useful for software engineers who develop software for FADEC and for testing engineers.

Key words: *FADEC, ECU, control system, gas turbine engine, digital regulator, software, software simulator, continuous integration, software testing, control system stability, AI-450M.*