УДК 004.4: 681.518.5

Разработка механизма тестирования модулей программ автоматизированных систем управления

Гришанова И.В.

Научно-производственное объединение автоматики им. академика Н.А.Семихатова, ул. Мамина-Сибиряка, 145, Екатеринбург, 620075, Россия

e-mail: alexndrovaiv@gmail.com

Аннотация: Рассматривается создание механизма автоматизированного тестирования, позволяющего контролировать ожидаемые и неожидаемые события. Проводится анализ двух вариантов тестирования программ: первый вариант – анализ протокола после выполнения программы; второй вариант – запуск фоновых задач параллельно с выполнением основной программы.

Разработанная система автоматического контроля выполнения программы позволяет проконтролировать любое требуемое событие и наглядно демонстрировать результат контроля.

Ключевые слова: тестирование, программный модуль, автоматический контроль, событие, фоновая задача.

Введение

В настоящее время автоматизированные системы управления применяются во многих сферах человеческой деятельности: промышленность, жилищно-коммунальные услуги, транспорт, авиация и космонавтика. К качеству их

реализации предъявляются очень высокие требования, так как неправильное функционирование этих систем может повлечь за собой серьезные последствия с риском для жизни людей и/или с большими финансовыми потерями. Поэтому до ввода в эксплуатацию АСУ очень важно проверить программное обеспечение (ПО) на соответствие предъявляемым требованиям, а также отработать реакцию АСУ на все события, которые могут возникнуть в процессе ее функционирования. Такие проверки называют тестированием. К тестированию существуют различные подходы. Вид тестирования выбирается исходя из требований [1], которым должно соответствовать ПО, и методов оценки показателей качества [2].В данной работе описан подход к тестированию отдельных программных модулей регистрационным методом. Задача: рассказать о концепции тестирования отдельных программных модулей на примере ПО наземной аппаратуры системы управления (НАСУ) для ракет «Союз-2» и о разработке механизма для создания этих тестов.

Исследование вариантов тестирования модулей программного обеспечения наземной аппаратуры системы управления для ракет «Союз-2» на автономном рабочем месте программиста

В НПОА проводится разработка ПО для НАСУ ракет «Союз-2», которое состоит из двух составляющих: системного и функционального ПО. Системное ПО представляет собой набор базовых задач, с помощью которых функциональные программисты реализуют ПО режимов наземной аппаратуры системы управления (далее по тексту – режимы). *Режим* – это набор проверок НАСУ и другой аппаратуры с ее помощью, которые необходимо провести перед запуском ракеты. В

каждом из режимов задействуется определенная аппаратура и для каждого существуют документы, в соответствии с которыми ведется разработка программ – *исходные данные* (ИД). ИД объемные, и для одного режима может использоваться не один такой документ. В них подробно описан каждый режим работы НАСУ: временные интервалы этапов, используемая аппаратура, условия для выполнения тех или иных операций, действия в нештатных ситуациях, документируемая информация. Как правило, программированием конкретного режима занимается определенный разработчик. Отработка программы происходит в несколько этапов; стоимость отработки программы на каждом следующем этапе возрастает. Следовательно, растет и стоимость ошибки. Разработанный механизм контроля хода режима позволяет определять ошибки в программе на начальном этапе отработки – на автономном рабочем месте программиста (АРМ-П), таким образом снижая затраты временных, аппаратных и человеческих ресурсов. Была поставлена задача создания системы автоматического контроля хода режимов, позволяющей повысить качество разрабатываемого ПО.

Любой тест направлен на выявление ошибок с целью последующего их устранения, следовательно, эффективным считается тест, который способен обнаруживать ошибки в программе. Тест, как и основную программу, создает человек, который не всегда полностью сконцентрирован при выполнении своей задачи. О психологических аспектах при тестировании пишет Майерс в книге «Искусство тестирования программ» [3], в которой он обозначает принципы тестирования. Озвучим те, которые легли в основу нашей концепции тестирования:

1) Следует избегать тестирования программы ее автором;

2) Тесты для неправильных и непредусмотренных входных данных следует разрабатывать так же тщательно, как для правильных и предусмотренных;

3) Нельзя планировать тестирование в предположении, что ошибки не будут обнаружены;

4) Необходимо проверять не только, делает ли программа то, для чего она предназначена, но и не делает ли она то, что не должна делать.

Исходя из первого принципа, было решено, что для эффективности тестирования необходимы три стороны:

1) разработчик программы (объекта тестирования);

2) тестировщик программы – тот, кто напишет программу-тест и с помощью нее проверит корректность работы объекта тестирования;

3) постановщик задач (разработчик ИД) – тот, кто оценит корректность работы обеих программ.

Прежде, чем создавать механизм тестирования, нужно было ознакомиться с классификацией тестов [4]и определить, какие виды тестов [5] будут проводиться. В ходе исследования была изучена классификация видов тестов по следующим признакам: по объекту тестирования; по знанию системы; по степени автоматизации; по степени изолированности компонентов; по времени проведения тестирования; по признаку позитивности сценариев; по степени подготовленности к

тестированию и по необходимости исполнения программного кода. В соответствии с этой классификацией тестирование, которое производится с помощью разработанного механизма, характеризуется следующим образом: позитивное и негативное динамическое автоматизированное функциональное модульное альфа-тестирование черного ящика по документации при приемке, проверке новой функциональности или для выявления регрессионных ошибок.

Разработанные средства тестирования предназначены для использования программистами режимов. При создании программ они используют специфический язык программирования. При отладке производится запуск режима на АРМ-П и визуально контролируются события: анимация и вывод сообщений в окно диагностики. Назовем эти действия прогоном. Затем после прогона смотрится *протокол режима*, в котором по порядку в строках записаны произошедшие события. С этим документом нужно было работать.

Было проанализировано два варианта решения поставленной задачи:

1) анализ протокола после выполнения программы;

2) запуск фоновых задач параллельно с выполнением основной программы.

Рассмотрим первый способ. Для этого потребуется программа, которая позволит создавать расписания и запускать тест определенного режима. Схема процесса тестирования с помощью такой программы показана на рисунке 1.

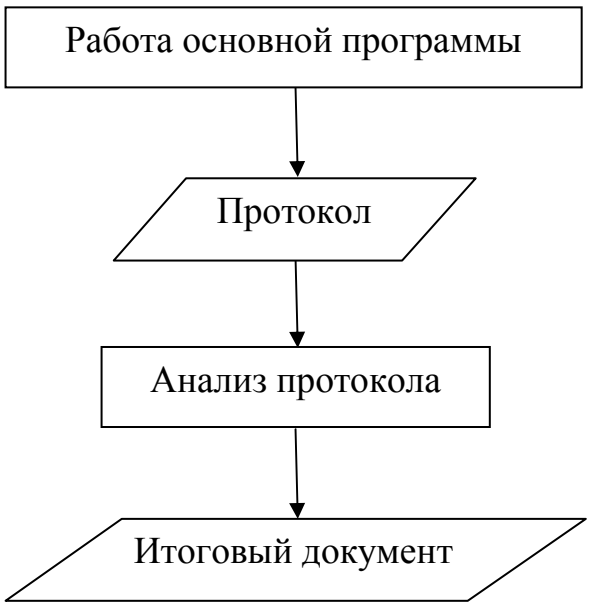


Рис. 1. Взаимодействие анализатора протокола с основной программой

Анализ протокола сводится к поиску строки в файле. На формирование результата будут влиять следующие параметры: время события и производимая операция. Операции между собой похожи мало: одни заключаются в выдаче данных, другие – в чтении; операция чтения может быть периодической; есть операции загрузки определенного файла, чтения из определенного устройства или записи в него. Чтобы оценить результат выполнения работы основной программы, нужно правильно сформировать условие для поиска строки в протоколе. Из-за большого числа различных критериев для сравнения это сделать трудно, а в некоторых случаях просто невозможно. Например, для операции периодического чтения: в протокол не ведется постоянная запись о том, что происходит чтение данных, следовательно, нельзя найти соответствующую строку, значит, нельзя сказать, произошло ли событие. Для того, чтобы необходимые для проверки строки появились в протоколе, нужно изменить исходный текст системной программы: в текст каждого контролируемого события вставить вывод строки-идентификатора

данного события.

Изначально предполагалось, что исходный текст системного ПО редактировать не придется. А в рассматриваемом варианте требуется редактирование исходного текста программы, следовательно, сложность создания механизма тестирования возрастает. Так как придется добавлять необходимые для анализа протокола данные, время выполнения программы и объем документируемой информации увеличится, следовательно, уменьшаются временная и пространственная эффективность [6].

Так же отдельное время потребуется на обработку полученной информации. Результаты анализа выполнения программы будут помещены в отдельный файл, и для оценки тестировщику нужно будет сравнить, по крайней мере, два документа с большим количеством строк. Этот процесс потребует большой концентрации внимания и не даст стопроцентной гарантии, что результат будет расшифрован правильно.

Можно поместить данные протокола выполнения основной программы и программы-теста в один документ. В данном случае, процедура визуального анализа протокола упростится, но снова снизится временная эффективность.

Существует еще один весомый недостаток такого метода тестирования: не все этапы режима ограничены временем, некоторые связаны с каким-либо событием. Следовательно, сформировать условие становится сложнее. Вполне возможно, что для анализа одного режима потребуется не один тест.

Средства тестирования получаются громоздкими, удобство их использования становится сомнительным, результат потребует дополнительного анализа. Данный метод тестирования имеет ряд серьезных недостатков. Но нельзя его отвергать, не проанализировав другие варианты автоматизации тестирования.

Теперь рассмотрим второй способ реализации средств тестирования: запуск фоновых задач параллельно с выполнением основной программы. Схема процесса тестирования этим способом показана на рисунке 2.

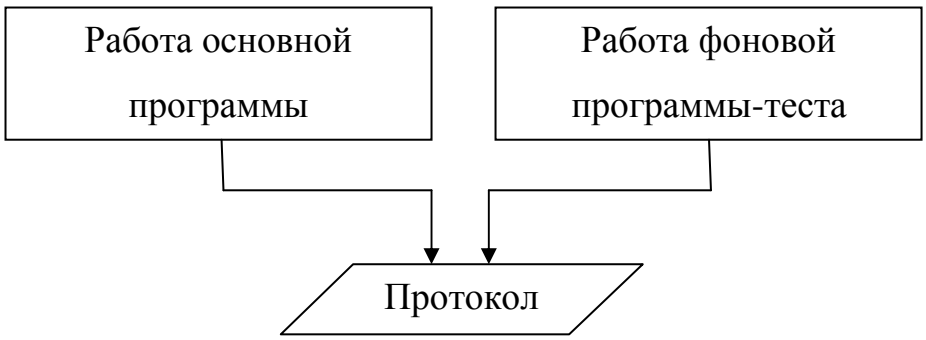


Рис. 2. Взаимодействие фоновых задач с основной программой

Возникает вопрос: не помешают ли тестовые задачи выполнению программы режима? Ответ: не помешают, так как при их запуске не будет происходить приостановка программы, а продолжат выполняться заданные действия, благодаря реализованному механизму с поддержкой параллельного выполнения кода различных программных модулей. Для организации фоновых задач потребуется редактирование исходных текстов системного ПО. Использование средств тестирования не будет отличаться от использования средств создания режимов, что облегчит задачу разработчику тестов.

Следующий вопрос: какие события можно будет контролировать с помощью

такого механизма? Ответ: любые. Для этого потребуется отредактировать участок исходного текста контролируемой задачи и создать задачу контроля.

Программа-тест будет запущена параллельно с проверяемой программой. Основными параметрами для контроля будут ожидаемые события и время, в течение которого они должны произойти. Результаты выполнения основной и фоновой программ будут занесены в один документ – это облегчит тестировщику оценку корректности работы поверяемой программы. Также не потребуется дополнительное время на программную обработку протокола, следовательно, временная эффективность данного способа тестирования выше, чем у предыдущего.

По результатам анализа составлена сравнительная таблица (табл. 1), в строках которой указаны критерии для сравнения двух предложенных способов тестирования. В этой таблице знаком «+» отмечена необходимость совершения указанных действий, а знаком «-» – отсутствие такой необходимости.

Преимущество варианта тестирования с помощью фоновых задач очевидно. Было принято решение использовать его для оценки соответствия программ режимов НАСУ требованиям ИД. Фоновые задачи называются задачами контроля хода режимов или просто задачами контроля. *Задачи контроля* хода режимов – это задачи, запускаемые программой-тестом для обнаружения требуемых событий или их отсутствия, а также неожидаемых событий в течение выполнения основной программы.

Таблица 1

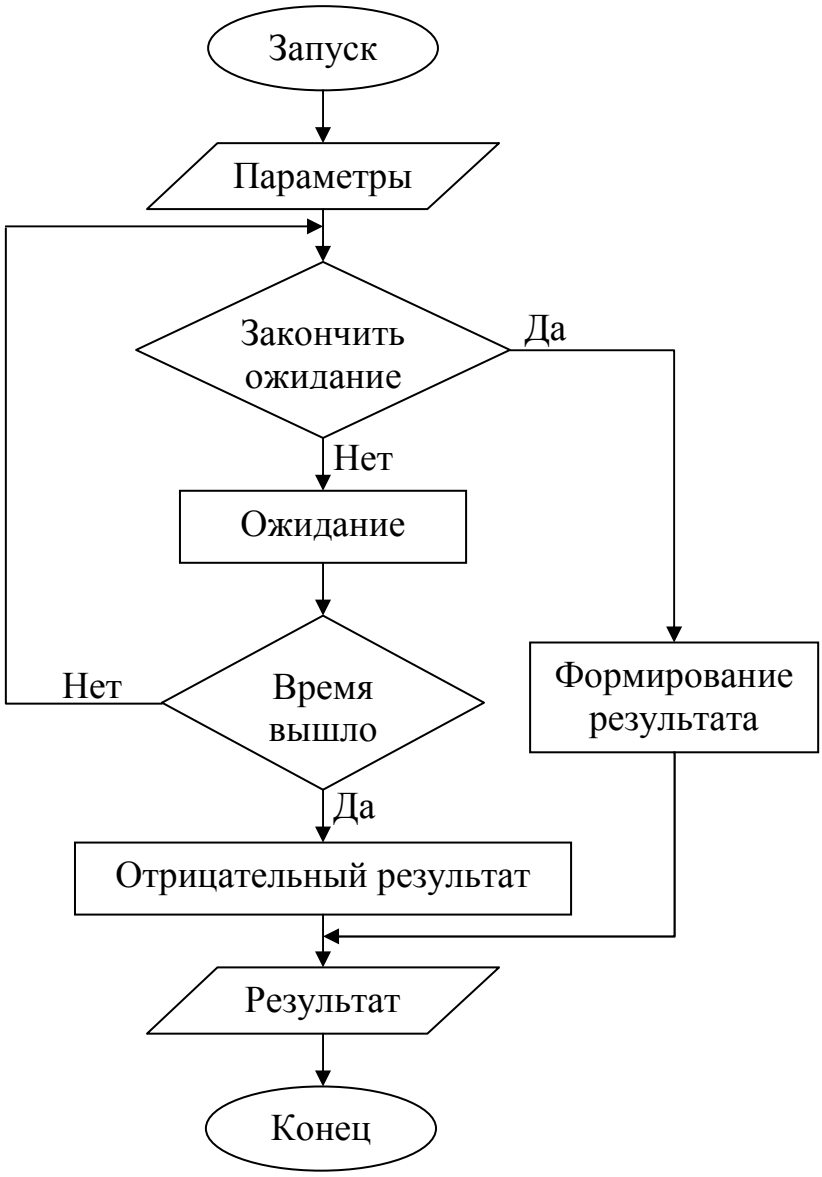
Сравнение вариантов тестирования

Параметр для сравнения	Анализ протокола	Фоновые задачи
Создание дополнительной среды разработки	+	-
Освоение тестировщиком новой среды	+	-
Редактирование исходного текста системного ПМО	+	+
Программная обработка данных после выполнения основной программы	+	-

Описание работы задач автоматического контроля

Работа задачи контроля представлена в виде схемы (рис. 3). Фоновые задачи контроля (рис. 3.а) запускаются программой-тестом. Каждая запущенная задача инициализируется и ждет сигнала от соответствующей задачи режима. Когда задача запускается в режиме, она посылает сигнал задаче контроля, что событие произошло. Задача контроля выходит из режима ожидания и сравнивает параметры задачи режима с заданными для контроля параметрами. Если они совпадают, задача формирует положительный результат и завершается (если нет дополнительных параметров), если нет – отрицательный и завершается.

а



б

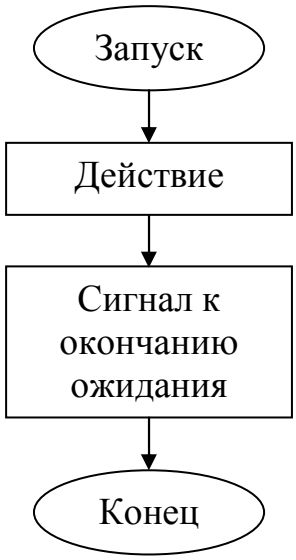


Рис. 3. Работа задач: а – фоновая задача контроля; б – задача программы режима

Такие действия программа выполняет, если на момент запуска задачи программы режима уже запущена и находится в процессе ожидания соответствующая задача контроля. Если же контроль не запущен, мы узнаем, что задача режима запущена несвоевременно. Для этого в задачу режима добавлен анализ, запущена ли соответствующая задача контроля. Если запущена, посылается сигнал окончания ожидания, а если не запущена, то в протокол заносится

сообщение о несвоевременно возникшем событии (рис. 4).

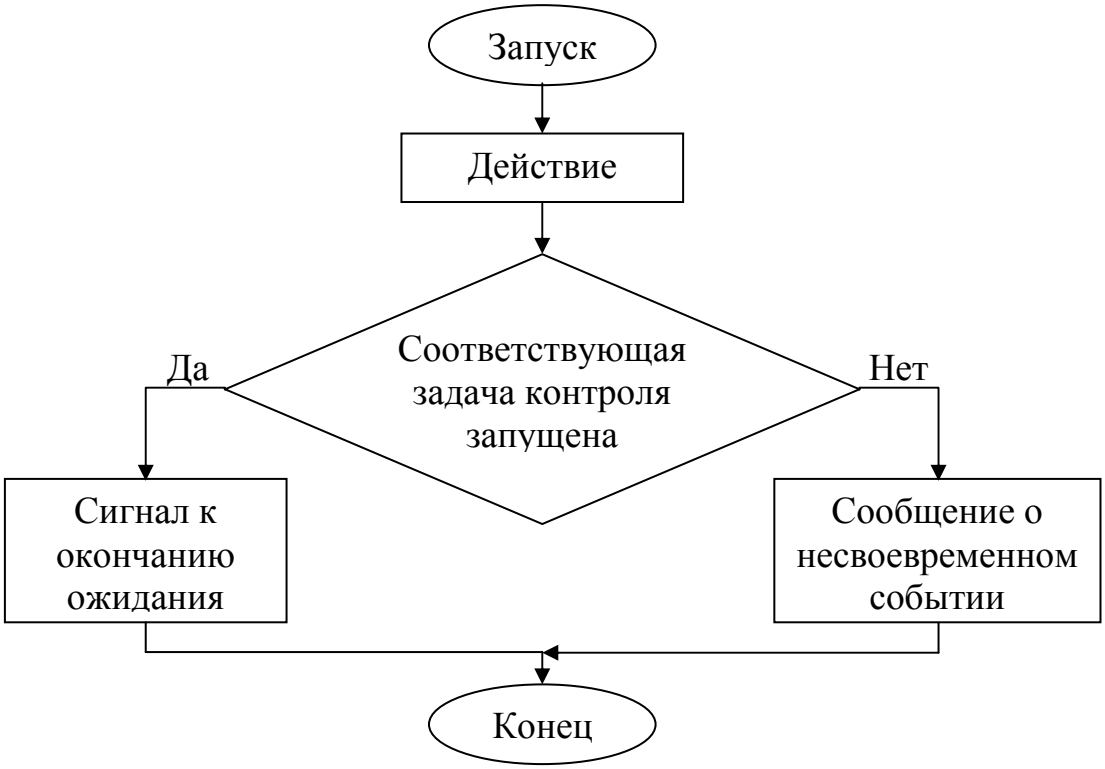


Рис. 4. Работа задачи программы режима

Перед тем, как приступать непосредственно к программированию задач контроля, были сформулированы требования к ним. Общие требования звучат так:

1) заносить в протокол режима положительные и отрицательные результаты контроля;

2) сообщения об успешном выполнении задачи должны быть раскрашены зеленым цветом;

3)сообщения об ошибках должны быть раскрашены красным цветом, чтобы они сразу выделялись;

4) помимо контроля хода режима по положительной ветке требуется контроль

по отрицательной ветке.

При разработке задач контроля эти требования были соблюдены, и можно получать подробную информацию о ходе режима, а затем анализировать ее по протоколу. Программа-тест является маской для программы режима или, можно еще сказать, с ее помощью задаются граничные условия. При попадании событий режима в отведенные им границы в протоколе будут сообщения зеленого цвета. Как только какое-либо событие режима выходит из указанных пределов, в протокол заносится сообщение об ошибке (красного цвета). Поскольку все диагностические сообщения регламентированы, легко расшифровать результаты прогона режима и найти участок программы, который не соответствует маске.

Стоит отметить, что тестирование производится на АРМ-П и диагностические сообщения заносятся в протокол. А в штатный протокол сообщения, относящиеся к тестам, попадать не должны. Эта задача решена с помощью использования флага: он позволяет включать и отключать компиляцию участков системного ПО, относящихся к тестированию. Таким образом, программа режима не зависит от того, тестируется ли она на АРМ-П или выполняется на штатной позиции.

Заключение

Разработанный механизм тестирования опробован и в настоящее время используется на нашем предприятии для контроля качества ПО НАСУ. В результате, с помощью созданного механизма разработчики ПО выявляют ошибки на начальном этапе отработки. Применение механизма тестирования позволило

выявить ошибки в ПО НАСУ, связанные с ошибочно заданным временем возникновения события; с отсутствием возникновения ожидаемого события; с возникновением неожидаемого события; с запуском ошибочного режима вместо требуемого. Также ошибки были выявлены в ИД: не учтены особенности программ, оговоренные в частных ИД.

Таким образом, применение разработанного механизма тестирования отдельных модулей ПО снижает затраты временных, аппаратных и человеческих ресурсов, что имеет важное экономическое значение для предприятия.

Библиографический список:

1. Леффингуэлл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. / Пер. с англ. Н.А.Орехова. М.: Издательский дом «Вильямс», 2002. – 448 с.

2. ГОСТ 28195-89 Оценка качества программных средств. Общие положения. М.: Издательство стандартов, 1989. – 38 с.

3. Майерс Г. Искусство тестирования программ. / Пер. с англ. Б.А. Позин. М.: Финансы и статистика, 1982. – 176 с.

4. Тестирование программного обеспечения. Протокол доступа к сетевому ресурсу: http://social.msdn.microsoft.com/Forums/ru-RU/e750a78b-0c1f-4766-81a2-7cea9b4b3ea2/-(дата создания: 12.10.2010)

5. Стотлемайер Д. Тестирование WEB-приложений: Средства и методы для автоматизированного и ручного тестирования программного обеспечения Web-сайтов. / Пер. с англ. Хахалин А. М.: КУДИЦ-ОБРАЗ, 2003. – 240 с.

6. Оценка программ. Протокол доступа к сетевому ресурсу: http://www.structur.h1.ru/ocenka.htm (дата обращения: 13.08.2013)