# Автоматизация функционального тестирования сложных программных средств

**А.А. Котлячков (МГТУ "Станкин")**

*Рассмотрены способы автоматизации тестирования сложных программных средств (СПС)[1]. Предложен вариант модели ЖЦ процесса тестирования. Проанализированы проблемы, связанные с автоматизацией тестирования, разработаны рекомендации по их решению.*

*Ключевые слова: сложные программные средства, автоматизация тестирования, функциональное и регрессионное тестирование.*

## Введение

Накопление знаний, опыта создания новых и применения готовых решений для разработки сложных программных средств (СПС) ведет к стремительному развитию сферы информационных технологий и ускорению темпов разработки программных продуктов. Высокие мировые стандарты качества диктуют необходимость организованного, контролируемого и методического отслеживания процессов разработки и контроля качества СПС. Развитие технологий и инструментов расширяет возможности работ по обеспечению качества методами автоматизированного тестирования и позволяет найти компромисс между требуемыми характеристиками качества создаваемой системы и ограниченностью производственных ресурсов.

Сокращение времени тестирования при разработке СПС – актуальная задача, которая стоит перед разработчиками ПО [1]. Существенно сократить время тестирования позволяет его автоматизация. Рассмотрим методологию оптимизации времени автоматизации тестирования.

Экономически целесообразно автоматизировать наиболее трудоемкие, повторяющиеся операции. Особенность сложных программных средств состоит в том, что они обладают большим объемом функциональности, следовательно, при тестировании наибольший объем работ приходится на функциональное тестирование. Поэтому далее речь пойдет только о функциональном тестировании программных продуктов для операционных систем, обладающих интерфейсом, например MS Windows или Mac OS.

## Терминология

Сформулируем основные понятия, используемые в статье, руководствуясь стандартом ISO/IEC TR 19759:2005. Совокупность знаний о разработке программного обеспечения.

*Таблица. Соответствие процессов ЖЦ ПС и ЖЦ процесса тестирования*

| Процессы ЖЦ ПС | Подпроцессы тестирования |
|---|---|
| Закупка<br>Поставка | Анализ требований |
| Разработка | Разработка новых тестов |
| Эксплуатация | Функциональное тестирование.<br>Регрессионное тестирование.<br>Подведение итогов |
| Сопровождение | Модификация тестов |

*Функциональное тестирование* – в широком смысле это тестирование функций ПО, проверка возможности правильного выполнения тех действий, которые впоследствии будет иметь возможность выполнять пользователь. В статье под функциональным тестированием подразумевается выполнение коротких тестов, затрагивающих одну или несколько новых или модифицированных функций СПС.

*Регрессионное тестирование* – проверка функций, внедренных в предыдущих версиях программы. Проверка функций, не проверяемых при функциональном тестировании.

*Скрипт* – набор команд тестового робота, выполнение которых симулирует работу пользователя.

*Рабочая версия программы* – в статье любая сборка ПО, позволяющая использовать основные функции.

*Тестовая версия программы* – сборка ПО, создаваемая для тестирования.

*Объект интерфейса* – любая деталь интерфейса (пользовательского GUI или программного API) ПО, с которой может взаимодействовать пользователь.

## Создание модели тестирования

Международный стандарт ISO 12207 описывает основные процессы и подпроцессы модели жизненного цикла (ЖЦ) программной системы (ПС), отражающей преимущества современных методов работы по ускоренной разработке приложений. Международный стандарт ISO 19759 описывает область знаний "тестирование программ" – детализация подпроцесса тестирования, цели, объекты и техники тестирования, методы оценки результатов тестирования.

Замена основных процессов ЖЦ ПС [2] аналогичными составляющими процесса тестирования [3] (таблица) позволяет получить модель, иллюстрирующую ЖЦ тестирования, учитывающую современные тенденции раннего подключения отдела обеспечения качества к процессу разработки программного продукта (рис. 1).

## Описание модели тестирования

Процесс тестирования состоит из основных этапов: анализа требований, разработки и выполнения тестов, документирования (протоколирования результатов), модификации тестов. Следует отметить, что тестирование – это многократно повторяемый

[1] *Сложное программное средство (СПС)* – ПО, имеющее обширную архитектуру, включающее множество вспомогательных подсистем, большой объем программного кода, развитую функциональность, сложную логику.

процесс, где каждой новой итерации соответствует новая тестовая или рабочая версия разрабатываемого программного продукта, содержащая некоторые изменения. Цель каждой итерации – проверка корректности внесенных изменений и корректности работы ПО в целом. Под изменением понимается внедренная новая функциональность, модифицированная ранее существующая функциональность или исправленная ошибка.

Процесс тестирования инициализируется, когда в отдел тестирования передается версия документации, соответствующая текущим изменениям, вносимым в разрабатываемое ПО. Модернизация, модификация, внедрение новых программных модулей и исправление ранее найденных ошибок, то есть изменение программного кода приводят к необходимости выполнения новой итерации тестирования. Внедрение новой функциональности требует пересмотра или разработки нового набора тестовых сценариев, а также перепланирования времени выполнения и объема работ в пределах текущей итерации.

На *первом этапе* необходимо сформировать тестовые данные на основе документации. Требования к СПС необходимы для разработки структуры тестов, контрольного листа и формирования отчета о проделанной работе. Описание функциональности СПС обеспечивает понимание внесенных разработчиками изменений и позволяет создать нетривиальные тестовые случаи. Документация от разработчика позволяет более точно локализовать область тестирования, определить места с наибольшей вероятностью возникновения аномалий.

На *втором этапе* разрабатываются новые тесты для проверки новой функциональности, которая появятся в новой версии ПО. На основании входных данных разрабатываются тесты для функционального тестирования. При разработке тестов учитываются предыдущий экспертный опыт тестирования, знания структуры и архитектуры объекта тестирования, архив найденных ошибок, знание характерных особенностей работы конкретного разработчика.

На *третьем этапе* производится функциональное тестирование [4], которое начинается, когда получена тестовая сборка новой версии ПС, и производится в соответствии с тестами, разработанными на предыдущем этапе. Найденные в процессе тестирования ошибки фиксируются и заносятся в систему управления ошибками. В зависимости от сложности исправления, серьезности и критичности ошибки исправляются на данном этапе или откладываются для дальнейшего планирования разработки. При исправлении найденных на данном этапе ошибок разработчики могут выпускать новые версии ПС. В этом случае тестирование продолжается с новой версией программы. Кроме выполнения заготовленных функциональных тестов обычно проводится свободное тестирование, целью которого является исследование реакции ПС на нерегламентированные действия

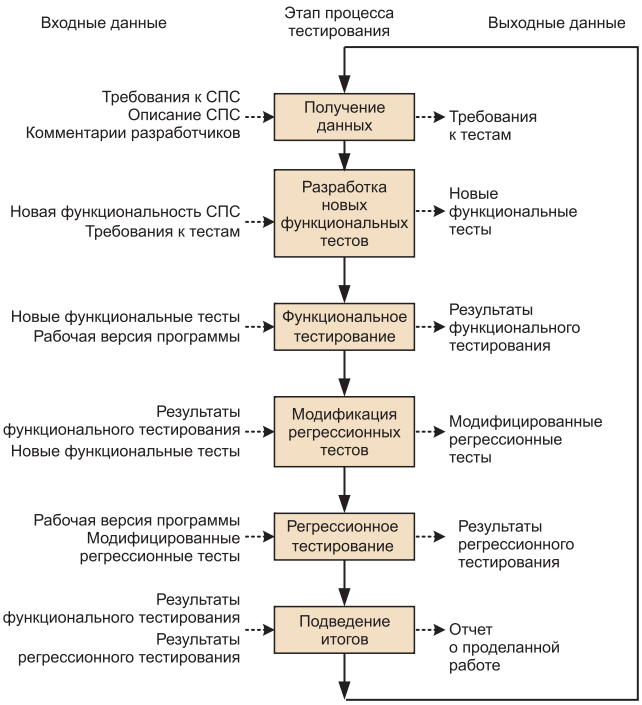


*Рис. 1. Модель процесса тестирования*

пользователя. Принимая во внимание факт, что поиск ошибок – это процесс практически бесконечный, требуется выбрать наиболее эффективный момент остановки тестирования. Работы по функциональному тестированию могут быть прекращены в момент, либо когда скорость нахождения ошибок существенно снижается, либо отводить на тестирование фиксированный период времени, адекватный числу внесенных изменений и количеству производственных ресурсов отдела тестирования.

На *четвертом этапе* выполняется регрессионное тестирование [4]. Проводится полная проверка функциональности ПС предыдущей базовой версии программы. Следует учитывать, что регрессионные тесты не учитывают новую функциональность ПС, которая может повлиять на ожидаемые результаты тестов. Цель регрессионного тестирования – проверить, что изменения не оказали незапланированного влияния на существующую функциональность. Возможно появление ранее исправленных и новых ошибок. В зависимости от сложности исправления, серьезности и критичности ошибки исправляются на данном этапе или откладываются для дальнейшего планирования разработки. При исправлении найденных на данном этапе ошибок разработчики могут выпускать базовые версии ПС.

На *пятом этапе* производится модификация регрессионных тестов. Новая версия регрессионных тестов должна учитывать все внесенные в ПС изменения. Это необходимо для выполнения регрессионного тестирования на следующей тестовой итерации. Для обновления регрессионных тестов используются функциональные тесты. Каждый функциональный тест обычно становится тестовым случаем регрессионного теста.

На *шестом этапе* подводятся итоги тестирования. По результатам тестирования анализируется число исправленных/неисправленных, новых ошибок, число удачно/неудачно внедренных изменений. Составляется отчет о проделанной работе. Принимается решение о работоспособности финальной версии ПС.

### Анализ модели тестирования

Модель тестирования позволяет проиллюстрировать процесс автоматизации тестирования СПС и выявить его недостатки. Автоматизация функциональных тестов осуществляется с помощью специализированного ПО – тестового робота [5], принцип работы которого заключается в эмуляции действий человека, то есть фактически роботу под контроль даются устройства ввода (клавиатура, мышь) и загружается программа действий. Функциональное автоматизированное тестирование (ФАТ) строится на умении робота различать тестовые объекты. Существуют два пути создания ФАТ – запись действий пользователя с помощью встроенного записывающего устройства или написание команд вручную, оперируя имеющимися в активе объектами и их свойствами. Существует возможность как последовательного выполнения действий, так и создания логических алгоритмов управления действиями. Для взаимодействия с интерфейсом ПС используется система распознавания объектов (кнопок, полей ввода, флагов и т.п.) [6]. Возможность распознавания объектов зависит от поддерживаемых роботом технологий (.Net, Java и т.д.). Умение отличать объекты друг от друга и запоминать зависит от способности распознавать свойства объекта (цвет, положение, тип, идентификатор, имя и др.). Свойства любого объекта можно классифицировать на постоянные и сессионные, уникальные и общие. Программировать робота на тестирование ПС можно, только если он способен опередить постоянные, уникальные свойства тестируемого объекта, которые не меняются со временем (при перезапуске ПС или ПК) и позволяют однозначно отличить данный объект от других. По причине динамического развития и большого числа технологий разработки ПО часто тестовые роботы не успевают развиваться, чтобы поддерживать новые и модифицирующиеся технологии.

Исходя из вышесказанного, понятно, что автоматизацию функционального тестирования можно осуществлять, лишь имея рабочую версию ПС. На рис. 2 на модели тестирования коричневым цветом показан период возможного автоматизированного тестирования (АТ) в рамках одной тестовой итерации. Серым цветом показаны этапы модели тестирования, приведенной на рис. 1. На рис.2 видно, что существует период, в течение которого работу по автоматизации вести невозможно по причине отсутствия рабочей версии ПС. Из-за ограниченности во времени автоматизируют регрессионные тесты, как самые трудоемкие и регулярно выполняемые. Такая автоматизация позволяет добиться существенного снижения времени выполнения работ по тестированию.

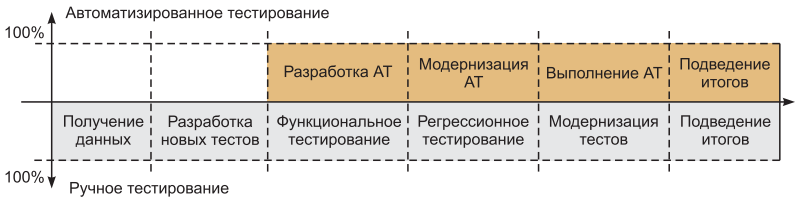


*Рис. 2. Период возможности автоматизации тестирования в течение тестовой итерации*

Этапы процесса АТ аналогичны этапам ручного тестирования за небольшим исключением. Регрессионное тестирование, в том числе автоматизированные скрипты, должны проверять ранее существующую функциональность. Но на каждой новой итерации тестирования версия тестируемого СПС может содержать изменения в интерфейсе и выполнение скриптов, привязанных к интерфейсу программы, будет нарушено. При появлении изменений интерфейса СПС скрипты требуют дополнительной отладки. Поэтому регрессионное АТ выполняется с учетом изменений в тестовой версии СПС.

Отметим, что время разработки и модификации АТ существенно дольше времени выполнения АТ [3]. Сокращение времени разработки (модификации) АТ позволит существенно уменьшить время, затрачиваемое на тестирование СПС в целом.

### Проблемы автоматизации тестирования

Можно сформулировать две основные проблемы АТ.

***Проблема 1.*** Взаимодействие робота с тестируемым приложением. Например, нужно выбрать заданное значение объекта "выпадающее меню"; распознанные свойства "текст строки" и действия "вниз", "вверх", "ввод". Проблема: невозможен выбор строки выпадающего меню по тексту.

Результат: взаимодействие с объектом затруднено. Проблему решают либо нахождением обходных путей – использованием других свойств объекта, либо нахождением аналогичных путей проверки функциональности или усложнением теста – внедрением логических алгоритмов, позволяющих оказать нужное воздействие на приложение и определением его реакции. Все роботы имеют встроенный язык программирования, расширяющий возможности оператора [6]. При наличии большого количества сложностей во взаимодействии робота и объекта тестирования возможен отказ от автоматизации и передача функциональности на ручное тестирование.

***Проблема 2.*** Ограниченность во времени. АТ – трудоемкая задача. Разработка скрипта занимает больше времени, чем разработка ручных тестов. Также необходима отладка скрипта на рабочей версии тестируемого СПС. На рис. 1 видно, что время, которое возможно отвести на отладку ФАТ на тестовой версии СПС, ограничено этапом 3. ФАТ рентабельно, если в

долгосрочной перспективе оно позволяет выполнять работу быстрее и дешевле, чем вручную. Сложность своевременной разработки автоматизированных тестов препятствует автоматизации функционального тестирования.

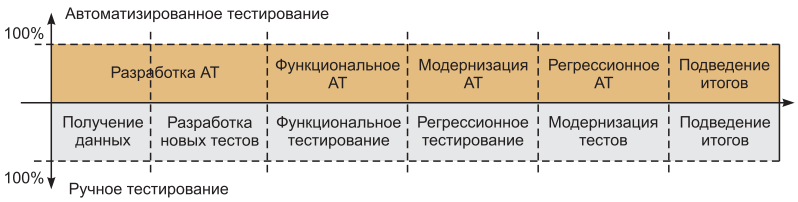


*Рис. 3. Увеличение периода возможности автоматизации тестирования в течение тестовой итерации*

## Решение проблем

Предлагается комплекс мероприятий, позволяющих в рамках ЖЦ процесса тестирования расширить временные границы интервала, в течение которого проводится АТ. Проблемы АТ являются следствием особенностей работы тестовых роботов.

На начальном этапе создания СПС можно организовать процесс разработки таким образом, который позволит начать АТ в самом начале процесса тестирования, не имея рабочей версии программного продукта. Достаточно знать имена, ключевые свойства и структуру расположения объектов в интерфейсе СПС. Это возможно в случае строго регламентированного процесса проектирования, когда кодирование ведется в близком соответствии с проектировочной документацией, что и подразумевает концепция создания СПС.

Уникальные имена элементов интерфейса, с которыми может взаимодействовать пользователь, можно формировать на этапе написания документации, описывающей структуру интерфейса СПС. В зависимости от используемой при разработке СПС технологии необходимо сформировать перечень свойств, которыми будет оперировать тестовый робот.

Заранее определить потенциально возможные технологии разработки и выбрать средство автоматизации, поддерживающее данные технологии, при использовании в разработке нестандартных компонент, проверять возможность их взаимодействия с роботом. Например, робот не может воздействовать (или воздействие затруднено) на ключевой объект программы, что осложняет весь процесс автоматизации и тестирования. Вполне возможно выгоднее и целесообразнее использовать другой компонент.

*Методология работы.* На этапе 1 дополнительно к документации необходимо прилагать изображения интерфейса из проектировочной документации с обозначенными новыми/измененными объектами с указанием необходимых свойств – уникального идентификатора и свойств, которые изменяются под действием пользователя. Также необходимо полное описание функциональности, действующих ограничений, а если функциональность сложная или скрытая – описание принципов работы и вариантов предполагаемого воздействия пользователя. На этапе 2 тестировщики, не имея рабочей версии тестируемого приложения, могут начать АТ. Таким образом, время разработки ФАТ увеличивается (рис. 3).

На этапе 2 возможно автоматизировать функциональные тесты, создав тем самым новые модули для автоматизированных регрессионных тестов. Это позволит на этапе 5 ускорить процесс модификации и адаптации существующих функциональных тестов под новую версию СПС.

Использование заранее запланированных технологий, объектов интерфейса с их именами и ключевыми свойствами позволяет решить проблему взаимодействия тестового робота с приложением.

Одной из основных причин возникновения ошибок является человеческий фактор [3]. Оптимизация обработки информации, необходимой человеку для работы – ключевой способ снижения влияния человеческого фактора на разработку программного продукта. СПС может обладать сложным интерфейсом, вследствие чего оперирование объектами при АТ будет затруднено. Для облегчения восприятия и структуризации информации о предмете тестирования предлагается использование матрицы функциональности.

Объект интерфейса характеризуется тремя основными свойствами [5]: местоположение, функциональность, уникальный идентификатор. Стандарты разработки интерфейса требуют использования глубины вложенности дочерних окон не более двух уровней. Таким образом, каждый объект интерфейса можно представить как элемент матрицы $A[x,y,z]$, где $A$ – параметр, идентифицирующий функциональность или позволяющий любым другим способом объединять объекты в смысловые группы; $X$, $Y$ – идентификаторы материнского объекта, например, главного и модального окна; $Z$ – идентификатор дочерних объектов.

Такой принцип хранения информации аналогичен дереву-каталогу, используемому для представления файлов на жестком диске в MS Windows. Он обеспечивает строго формализованную связь между сущностями процесса разработки [1]: тестовым планом, тестами, системой управления ошибок, планом работ по разработки СПС и документацией по разработке, например, требованиями и вариантами использования (рис. 4).

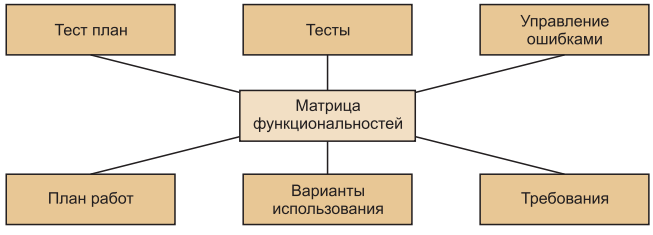


*Рис. 4. Связь матрицы функциональности с сущностями процесса разработки*

*Методология работы.* Идентификаторы объектов создаются при написании требований к СПС в разделе описания интерфейса. Использование идентификаторов объектов в тест-плане и плане работ позволяет точно определить спектр и объем работ. В тестах идентификаторы используются для однозначного описания действий тестировщика. В системе управления ошибок идентификаторы позволяют однозначно описать порядок действий, необходимых для воспроизведения ошибки, и позволяют проводить статистический анализ ошибок с учетом локализации. Применение идентификаторов в вариантах использования позволяет точно описать запланированные действия пользователя, что в последствии упрощает разработку и автоматизацию тестов, так как фактически содержит элементы скриптов.

Формализация отношений между сущностями и численная идентификация объектов позволяет перейти к строгой количественной оценке характеристик качества процесса разработки: по объему работ тестирования и АТ, по тестовому покрытию, проценту автоматизации, пересечению тестов в отношении функциональности, планированию тестирования, индикации процесса выполнения тестирования, проценту ошибок и их локализации.

## Заключение

Таким образом, достоинствами построенной модели тестирования являются:

- создание ФАТ на раннем этапе ЖЦ тестирования ПО;
- выделение большего времени на разработку ФАТ;
- обеспечивает взаимодействие между группами разработки и тестирования;
- возможность точной фиксации места возникновения дефекта (с точки зрения тестирования);
- однозначность определения объектов интерфейса;
- модульность разработки тестов;
- ускорение модификации автоматизированного регрессионного тестирования;
- снижение трудозатрат на тестирование в целом.

К недостаткам можно отнести необходимость внесения изменений в традиционный процесс разработки ПО, что может быть негативно воспринято разработчиками.

Перспективным направлением ускорения разработки СПС является интегрирование инструментов тестирования в среду разработки, что позволяет упростить процесс передачи необходимой технологической информации от разработчиков к тестировщикам и облегчает реализацию предложенной модели тестирования.

## Список литературы


1. *Липаев В.В.* Программная инженерия. Методологические основы. Учебник. М.: ТЕИС. 2006.
2. *Липаев В.В.* Процессы и стандарты жизненного цикла сложных программных средств. Справочник. М.: Синтег. 2006.
3. *Блэк Р.* Ключевые процессы тестирования. М.: Лори. 2006.
4. *Бейзер Б.* Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. СПб.: Питер 2004.
5. *Дастин Э., Решка Дж., Пол Дж.* Автоматизированное тестирование программного обеспечения. М.: Лори. 2003.
6. *Винниченко И.* Автоматизация процессов тестирования. СПб.: Питер 2004.



***Котлячков Александр Алексеевич*** *– старший инженер-программист кафедры "Информационные системы" МГТУ "Станкин".*

*Контактный телефон (499) 973-94-27. E-mail: nuclear_fly@mail.ru*


## Компактный EtherCAT-сервопривод на 40 А

Семейство высокодинамичных и гибких EtherCAT-сервоприводов Beckhoff пополнилось новой моделью AX5140, рассчитанной на номинальный ток 40 А и закрывающей промежуток между версиями на 25 А и 60 А в рамках успешной серии AX5000. Отличительной чертой AX5140 является ее высокая производительность при очень малых размерах устройства.

Пополнение семейства EtherCAT-сервоприводов AX51xx версией с номинальным выходным током 40 А означает, что одноканальные приводы этой серии номиналом 1,5...170 А становятся еще более масштабируемыми по производительности. Гибкая модель AX5140 рассчитана на работу в трехфазной сети с напряжением ~100...480 В. Этот компактный сервопривод можно успешно использовать с оборудованием для обработки пластмасс, обработки металла, фрезеровальными и сверлильными станками, экструдерами и т.д.

Идентичная по конструкции версиям на 18 А и 25 А модель AX5140 представляет собой исключительно компактное устройство для своего класса производительности. Благодаря интеграции сетевых фильтров и балластного резистора, а также использованию высокоэффективного радиатора ее размеры удалось свести до минимума - 300x185x232 мм (ВхШхГ). Для создания высокопроизводительной многоосевой системы модель AX5140 с помощью моста AX-Bridge можно легко и быстро комбинировать с дополнительными приводами AX5000. Набор принадлежностей AX-Bridge включает блок питания, промежуточный контур и схему управляющего и тормозного напряжения =24 В.

Модель AX5140 имеет те же алгоритмы и структуры контроллера, что и другие устройства серии AX5100, и исключительно гибка в отношении подключения двигателей. Например, в сочетании с синхронными серводвигателями AM307x с кодом обмотки Q она превращается в высокодинамичный приводной модуль. Дополнительные особенности, оставшиеся без изменений, включают оборудование с интерфейсом множественной обратной связи, экран диагностики и настройки, а также интерфейс ввода/вывода дискретных сигналов. Благодаря системе Beckhoff TwinSAFE в EtherCAT-сервоприводы встроены также функции противоаварийной защиты, такие как блокировка перезапуска.

*Контактный телефон (495) 981-64-54. E-mail: russia@beckhoff.com Http://www.beckhoff.ru*