ISP RAS

CTESK

Автоматизированное тестирование приложений на языке C

Проблема тестирования

В современных проектах по разработке программного обеспечения стоимость тестирования может составлять более половины бюджета. Более того, нет ничего необычного, если стоимость поддержки продукта после выпуска превышает стоимость его разработки. В то же время, эффективность тестирования не слишком высока. Всё это ведет к неприятным экономическим потерям, а в некоторых случаях и к катастрофам с недопустимыми последствиями.

Основной причиной такой ситуации является увеличение размеров и сложности программного обеспечения при одновременном сокращении времени выхода на рынок. Чтобы соответствовать требованиям современной ситуации, тестирование ПО нуждается в более систематических и гибких подходах, большей автоматизации и более высокой доле повторно используемых компонентов.

CTESK предлагает такой подход и экономически эффективный способ автоматизации важного вида тестирования — *функционального тестирования*, которое также называют *тестирование соответствия*. Более подробно это понятие рассмотрено в следующем разделе. Инструмент CTESK позволяет создавать больше тестов более высокго качества с меньшими затратами. Автоматизированное тестирование часто связывают с инструментами для автоматического выполнения тестов, написанных вручную. CTESK не относится к таким инструментам. Он позволяет автоматизировать и создание, и выполнение тестов.

Функциональное тестирование

Существуют различные виды тестирования: тестирование производительности, тестирование в предельных режимах и т.п. Но вне зависимости от того, удовлетворяет ли реализация требованиям по производительности и по потребляемым ресурсам, её поведение должно быть корректным.. Другими словами, поведение системы должно *соответствовать функциональным требованиям*.

Взаимодействие программы с внешним миром осуществляется через некоторый интерфейс. Например, это может быть программный интерфейс (API) программного компонента. Функциональные требования не описывают реализацию системы. Они определяют *внешнее наблюдаемое поведение* системы в процессе взаимодействия с окружением в терминах *интерфейса* системы.

Поведение системы соответствует функциональным требованиям, если любой наблюдаемый эффект её работы допускается функциональными требованиями. Таким образом, целью функционального тестирования является проверка того, что реализация системы делает то, что должна, и не делает того, чего не должна. Функциональное тестирование должно выявлять систематические нарушения функциональных требований. Обычно такие нарушения являются следствием взаимонепонимания между заказчиками и разработчиками, между архитекторами и программистами, а также при изменении требований в процессе разработки. Возможно более раннее выявление таких ошибок чрезвычайно важно для сокращения риска провала при интеграции системы.

Функциональное тестирование является существенной частью работ по тестированию программного обеспечения, поэтому его автоматизация позволяет существенно снизить затраты на тестирование.

Подход CTESK

Автоматизация функционального тестирования возможна только в том случае, если функциональные требования заданы в строго формальном виде. *Формально* в данном случае означает представление, доступное для *компьютерного анализа* и имеющего *однозначную интерпретацию*. Это не обязательно связано со сложными математическими рассуждениями. Различие между неформальными и формальными спецификациями функциональных требований связано не с различием между языками программирования и математической нотацией, а с разницей между естественными языками и языками программирования.

CTESK работает с формальными спецификациями системы на основе утверждений о поведении в форме пред- и постусловий. Они определяются на спецификационном расширении языка C — SeC, компактном и достаточно понятном языке для опытных C-разработчиков.

Формальные спецификации определяют модель поведения системы в виде автомата состояний. Кроме того, спецификации содержат определения критериев покрытия спецификаций. Тестовая система CTESK автоматически контролирует достигнутый уровень тестового покрытия. Компоненты тестовой системы, описанные на языке SeC, транслируются в код на языке C, ктоорый проверяет корректность поведения тестируемой реализации относительно используемой модели.

Для автоматического построения *тестовой последовательности* (последовательности воздействий на тестируемую систему через её интерфейс) используются математические алгоритмы, реализованные *тестовым обходчиком*. Эти алгоритмы обеспечивают исследование огромного количества состояний системы и проверку её поведения в каждом из них для достижения необходимого уровня тестового покрытия. Разработчик тестов должен создать короткое описание тестового варианта: какую часть интерфейса системы он должен проверять, как следует выбирать параметры для этих интерфейсных вызовов и в каком тестовом состоянии их следует производить (тестовое состояние включает в себя историю взаимодействия системы с окружением во время тестирования).

Для описания тестовых вариантов на языке SeC разрабатываются *тестовые сценарии*, каждый из которых описывает параметризованный набор тестовых вариантов.

Взаимодействие тестовой системы с тестируемой реализацией осуществляется через разрабатываемые на языке SeC *тестовые медиаторы*. CTESK транслятор транслирует тестовые медиаторы в код на языке C.

Интеграция CTESK в процесс разработки программного обеспечения

Обычно разработка программного обеспечения начинается со сбора и анализа требований, которые затем преобразуются в описание архитектуры системы и наборы требований к отдельным компонентам. На основе требований разрабатывается реализация компонентов. Затем поведение отдельных компонентов и их взаимодействие во время модульного и интеграционного тестирования проверяется на соответствие требованиям.

Для автоматизации тестирования с использованием CTESK, требования к программному обеспечению должны быть формальными в указанном ранее смысле. Формальные спецификации, необходимые для автоматизации тестирования с помощью CTESK, полезны и для других целей. Строгость формальных спецификаций заставляет разработчика продумывать их более тщательно. Однозначное толкование спецификаций устраняет противоречия, неоднозначности и взаимонепонимание между разработчиками. Таким образом, создание формальных спецификаций помогает обнаруживать ошибки раньше, чем при использовании традиционного процесса разработки.

Следовательно, наилучший способ использования CTESK в процессе разработки — не вводить дополнительную отдельную деятельность по тестированию с помощью CTESK, а использовать формальные спецификации на ранних стадиях разработки, автоматизировать разработку модульных и интеграционных тестов одновременно с проектированием и реализацией системы. Рисунок ниже иллюстрирует этот подход.

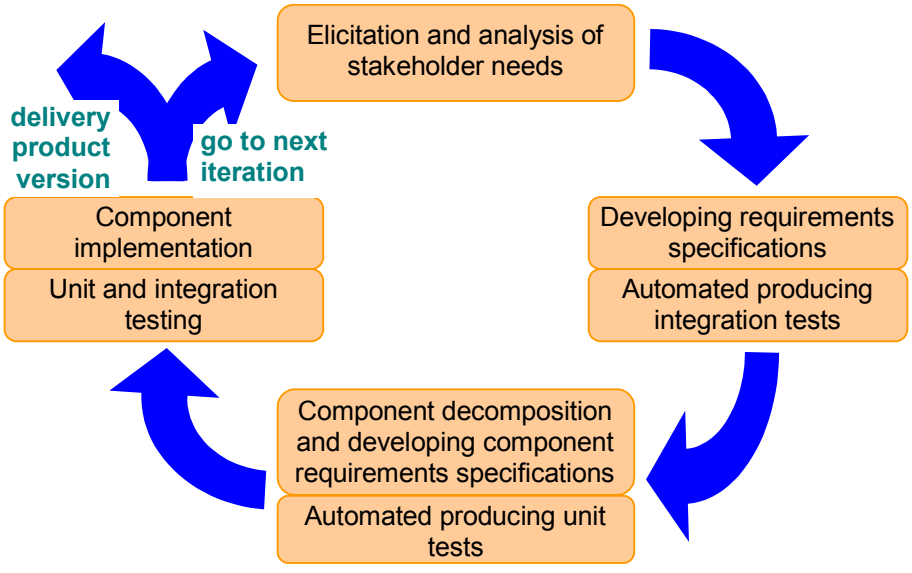

Рис. 1 Интеграция CTESK в процесс разработки программного обеспечения

Однако использование формальных спецификаций в процессе разработки программного обеспечения часто сталкивается с различными трудностями в виде обычаев и традиций, корпоративных правил разработки, нехватки времени или недостатка квалификации. В этом случае целесообразно использовать CTESK локально. CTESK может быть использован для автоматизации модульного тестирования одного или нескольких наиболее важных компонентов, или только для интеграционного тестирования. CTESK подходит и для использования в проектах обратной инженерии. Формальные спецификации могут определять высокоуровневую модель поведения системы, независимую от деталей реализации. С помощью тестовых медиаторов тесты на соответствие этой модели могут быть использованы для различных реализаций. То есть, CTESK предлагает хорошую поддержку автоматизированной разработки регрессионных тестов и автоматизированного регрессионного тестирования.

Вышеуказанные пути интеграции CTESK в процесс разработки могут быть применены в пилотных проектах. Если пилотные проекты покажут, что использование формальных спецификаций приносит ощутимую выгоду, их использование может быть расширено за пределы области тестирования.

Основные возможности и архитектура тестового набора

CTESK предоставляет каркас для разработки системы автоматизированного функционального тестирования на основе формальных спецификаций. Его основные возможности:

- Спецификация требований на языке SeC — компактном и понятном спецификационном расширении языка программирования C. Результатом является высокоуровневая модель поведения тестируемой системы, по которой автоматически генерируются тестовые оракулы.
- Метрики тестового покрытия, основанные на спецификации — предоставляют средства измерения тестового покрытия и разумный критерий окончания тестирования.
- Определение тестовых сценариев, описывающих тестовые варианты.
- Автоматическая систематическая генерация тестовых последовательностей во время выполнения теста.
- Гибкое связывание тестовой системы с тестируемой реализацией посредством тестовых медиаторов.
- Поддержка тестирования распределенных систем и систем с асинхронным интерфейсом.
- Автоматическое выполнение тестов.
- Генерация отчетов о достигнутом тестовом покрытии и об обнаруженных ошибках.

Общая архитектура автоматизированного тестирования с использованием CTESK показана на следующем рисунке.

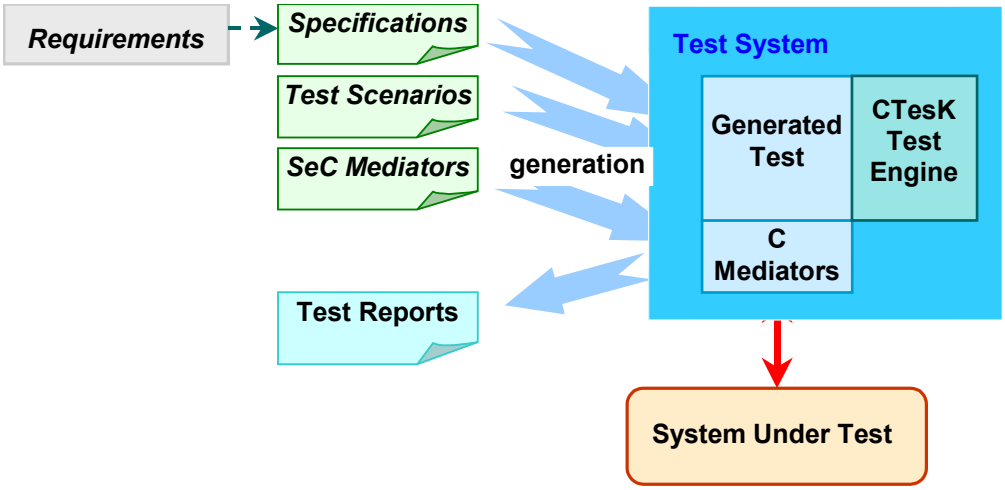

Рис. 2 Общая архитектура автоматизированноготестированияс использованием CTESK

С использованием CTESK может быть реализован следующий процесс автоматизированного функционального тестирования:

1. **Решение об области и глубине тестирования.**
   На этом шаге выделяется набор компонентов целевой системы, которые будут тестироваться с помощью CTESK. Для каждого компонента устанавливается его интерфейс и основные требования, которые определяют уровень абстракции модели компонента.
2. **Разработка модели поведения выделенных компонентов целевой системы.**
   Когда набор компонентов выделен, для каждого выделенного компонента целевой системы разрабатываются формальные спецификации поведения.
3. **Определение целей тестирования.**
   На этом шаге устанавливаются цели тестирования в виде уровня тестового покрытия спецификаций и исходного кода выбранных компонентов. Его называют *целевым уровнем покрытия*. Затем разрабатываются тестовые варианты, необходимые для достижения целевого уровня покрытия. Эти тестовые варианты могут быть разных типов; они могут проверять разные аспекты поведения системы и могут быть по-разному устроены. Также определяются компоненты целевой системы и их интерфейсы, используемые тестовыми вариантами.
4. **Разработка тестовых медиаторов.**
   Далее на языке SeC разрабатываются тестовые медиаторы для каждого тестируемого компонента. Медиатор связывает модель с тестируемой реализацией, функциональность которой описывается моделью. Один и тот же медиатор может быть использован несколькими тестовыми вариантами.
5. **Разработка тестового сценария.**
   После разработки набора тестовых вариантов разрабатывается тестовый сценарий. Он определяет обобщенное состояние набора выбранных компонентов целевой системы и тестовые воздействия, которые необходимо оказать на эти компоненты в каждом состоянии. В сущности, тестовый сценарий неявно описывает конечный автомат по которому генерируются тестовые последовательности.
6. **Отладка тестового набора.**
   После разработки всех необходимых компонентов тестового набора производится их интеграция, подключение к тестируемой реализации и отладка. Ошибки,

найденные в тестовой системе (не в целевой системе!) должны быть исправлены до перехода к следующему шагу.

7. **Выполнение тестов и анализ результатов.**
   Готовый тестовый набор выполняется. По результатам выполнения создается отчет о найденных ошибках и достигнутом уровне тестового покрытия.

После выполнения тестов и анализа отчета о достигнутом уровне тестового покрытия может понадобиться разработка дополнительных тестов, покрывающих недостающее.

Процесс тестирования назван «автоматизированным» потому что он осуществляет:

- генерацию компонентов тестового набора из спецификаций и описания тестового сценария;
- генерацию тестовых последовательностей:
- выполнение тестов;
- генерацию отчета об ошибках и достигнутом покрытии.

Области применения

Фундаментальных ограничений для применения CTESK нет. CTESK хорошо подходит для тестирования любых компонентов с программным интерфейсом (API). В других случах, таких как тестирование распределенных систем и систем с асинхронным интерфейсом (например, основанных на передаче сообщений) используются специальные конструкции языка SeC для описания поведения целевой системы и создания релевантного тестового сценария. Тестовая система автоматически проверяет корректность поведения реализации в случае её асинхронного и/или параллельного взаимодействия с окружением.

Однако, инструменты подобные CTESK обычно применяются при разработке программного обеспечения, к надежности которого предъявляются повышенные требования. Поэтому наиболее перспективными областями применения CTESK являются:

- Критически важное программное обеспечение (системы управления в авиации, охранные системы, встроенное программное обеспечение, системы контроля промышленного производства, медицинские системы).
- Системное программное обеспечение, такое как компоненты операционной системы, веб-сервисы.
- Телекоммуникационное программное обеспечение.
- Реализации стандартов (например, реализации протоколов, языков и интернет-приложения).

Преимущества

Наиболее значительные преимуществами использования CTESK таковы:

- **Автоматизация создания тестов**

  Транслятор CTESK позволяет получать тесты автоматически по спецификациями требований и тестовым сценариям, которые разрабатываются на языке SeC —

компактном и понятном спецификационном расширении языка C. Пользователю не приходится изучать отдельный спецификационный язык.

- **Автоматическое вынесение вердикта.**

CTESK **автоматически выносит вердикт** о соответствии реального поведения системы спецификациям. Этот вердикт основан на анализе результатов тестирования.

- **Автоматическая генерация тестовых последовательностей**

По относительно небольшим и простым описаниям тестовых сценариев и спецификации целевой системы CTESK используя математические алгоритмы автоматически генерирует эффективные тесты. Эти тесты проверяют поведение системы во всех существенно различных состояниях, определенных спецификацией. Поэтому отпадает необходимость в тестовых скриптах, которые составляют большую часть традиционного тестового набора.

- **Тестирование асинхронных и параллельных интерфейсов**

Язык SeC предоставляет специальные возможности для описания систем с асинхронным интерфейсом (например, основанных на передаче сообщений) и создания тестовых сценариев с параллельными взаимодействиями (синхронными или асинхронными) между системой и её окружением. Тестовая система автоматически проверяет корректность поведения системы в случае таких взаимодействий.

- **Повышение качества тестирования**

Тестовые последовательности, генерируемые CTESK, систематичеки исследуют все существенно различные состояния целевой системы . Это позволяет добиться высокого уровня тестового покрытия.

- **Поддержка регрессионного тестирования**

CTESK полностью поддерживает регрессионное тестирование. Гибкий механизм связывания спецификаций и реализайии позволяет проверить поведение новой версии реализации на соответствие любой предыдущей версии спецификации.

- **Высокая степень повторного использования спецификаций и других компонентов тестового набора, разрабатываемых вручную**

Использование моделей высокого уровня и независимость спецификаций от реализации обеспечивают высокую степень повторного использования спецификаций и других компонентов тестового набора, разрабатываемых вручную.

- **Общее улучшение процесса разработки программного обеспечения**

Кроме улучшения процесса тестирования, использование CTESK позволяет улучшить процесс разработки программного обеспечения в целом, позволяя разрабатывать тесты одновременно с реализацией, что сокращает общее время разработки.

- **Интеграция с существующими средами разработки для C/C++**

Для удобства разработки компонентов тестовой системы CTESK интегрирован в среду разработки Eclipse C/C++ Development Tools.

Системные требования

Существующая версия CTESK работает на Windows (с использованием cygwin) и на Linux платформах. Возможен перенос инструмента на другие платформы. Для работы CTESK необходимы Java Runtime Environment, Eclipse C/C++ Development Tools, gcc.