T2C: технология автоматизированной разработки тестов базовой функциональности программных интерфейсов

В. В. Рубанов, А. В. Хорошилов, Е. А. Шатохин
{vrub, khoroshilov, spectre}@ispras.ru

Аннотация. В статье обсуждается задача автоматизации разработки тестов базовой функциональности программных интерфейсов (API). Рассматриваются существующие решения по разработке таких тестов и описываются разработанные в ИСП РАН технология и инструментальные средства T2C, нацеленные на создание тестов «среднего» уровня по качеству тестирования и объему соответствующих трудозатрат. Приводится статистика результатов использования T2C в рамках проекта по тестированию интерфейсов системных библиотек Linux, описываемых стандартом LSB.

1. Введение

Хорошо известно, что проверить корректность работы любой серьезной программной системы во всех ситуациях, которые могут встретиться в ходе ее реальной эксплуатации, невозможно. Поэтому тестирование программного обеспечения неизбежно связано с поиском компромисса между тщательностью проверок и ограничениями на имеющиеся ресурсы.

Поиск оптимального решения зависит от множества факторов, определяющих специфику конкретного проекта. В данной работе мы рассмотрим разработку тестов, предназначенных для тестирования на соответствие требованиям стандарта программных интерфейсов на языке программирования Си. Такая постановка задачи предопределяет значение целого ряда факторов.

- Необходимость сопровождения тестов вместе с развитием стандарта.
- Наличие стандарта подразумевает существование достаточно качественного описания требований к функциональности тестируемой системы. Хотя для стандартов, находящихся в стадии активного развития, такое предположение не всегда оказывается верным.
- Потребность в простоте анализа обнаруживаемых ошибок, так как во многих случаях ошибки в реализации будут анализировать не разработчики тестов, а представители сторонних организаций, проверяющих свои продукты на соответствие требованиям стандарта.

Эти особенности формируют ключевые требования к результирующему тестовому набору, и тем самым, к подходу по его разработке. Но из них не следует никаких ограничений на возможные решения конфликта между детальностью тестирования и имеющимися ресурсами.

Наиболее распространенным решением при разработке тестов на соответствие является выбор тестирования базовой функциональности, под которым понимается тестирование, обеспечивающее проверку корректности функционирования системы на нескольких основных вариантах её использования, включающих в себя и некоторые ошибочные ситуации. Распространенность такого выбора объясняется тем, что тестирование базовой функциональности позволяет с минимальными затратами выявить большинство наиболее грубых отклонений от стандарта.

Существуют и альтернативные решения. В случаях, когда в короткие сроки требуется обеспечить покрытие очень большого количества функций, выбор падает на менее тщательное тестирование, например, такое, которое нацелено на проверку только минимальной работоспособности каждой функции. Одна из технологий разработки тестов такого вида — технология Azov — представлена в настоящем сборнике.

В ситуациях, когда обеспечение точного соответствия стандарту является крайне важным и подкрепляется наличием достаточного количества ресурсов, выбирается более тщательное тестирование, например, такое, которое нацелено на проверку всех выделенных классов тестовых ситуаций для каждого отдельного аспекта функциональности каждой интерфейсной функции. Примером инструментов, поддерживающих разработку столь детальных тестов, которые при этом еще и остаются удобными для дальнейшего сопровождения и развития, является набор инструментов CTESK [1]. Такие особенности CTESK как:

- наличие средств для формального описания требований к тестируемой системе;
- поддержка автоматической генерации последовательностей тестовых воздействий при помощи динамического построения модели поведения тестируемой системы;
- широкий набор возможностей для задания метрик качества тестирования в терминах модели требований с автоматизацией сбора информации о достигнутом покрытии;

помогают организовать систематический перебор всех тестовых ситуаций и упростить анализ тестового покрытия.

Набор инструментов T2C, о котором пойдет речь в настоящей статье, нацелен на разработку тестов базовой функциональности. Превосходя возможности технологии Azov и уступая инструментам CTESK по уровню тщательности тестирования получаемых тестов, инструменты T2C позволяют эффективно достигать середины в балансе между качеством тестирования и ресурсами,

необходимыми для построения соответствующих тестов. При этом инструменты T2C поддерживают основные рекомендации по работе с требованиями при разработке тестов на соответствие, такие как составление каталога элементарных требований стандарта, обеспечение прослеживаемости требований стандарта в тестах и измерение качества тестирования в терминах покрытия элементарных требований.

Данная статья построена следующим образом. В первом разделе мы рассмотрим ряд подходов, решающих близкие задачи, а также обсудим их достоинства и недостатки применительно к стоящим перед нами целям. Далее мы представим основные особенности инструментов T2C и поддерживаемого ими процесса разработки. Результаты будут проиллюстрированы на опыте применения данного подхода при разработке тестов на соответствие требованиям стандарта LSB [2] для ряда библиотек из стека GTK+ и fontconfig. В заключении мы обсудим возможные направления дальнейшего развития подхода и его интеграции с другими инструментами семейства UniTESK.

2. Средства разработки тестов базовой функциональности программных интерфейсов

2.1. MANUAL

Тестовые системы, обеспечивающие тщательное тестирование, как правило, требуют для своей работы целый ряд сервисов нижележащей операционной системы. Поэтому в случаях, когда объектом тестирования является сама операционная система, для запуска таких тестов требуется относительная стабильность объекта тестирования.

Для смягчения этой проблемы, а также для минимизации непреднамеренного воздействия тестовой системы на целевую систему часто применяется распределенная архитектура построения тестов, при которой для выполнения большей части задач используется вспомогательная инструментальная машина, а на тестируемой системе работает только небольшой тестовый агент. Но даже в таком случае, для взаимодействия тестового агента с инструментальной машиной необходима работоспособность отдельных компонентов тестируемой системы.

По этой причине, прежде чем приступать к тщательному тестированию программных интерфейсов целевой системы, необходимо удостовериться в работоспособности ее ключевых компонентов.

В проекте по тестированию POSIX-совместимой операционной системы реального времени для встраиваемых устройств, в отделе технологий программирования ИСП РАН был разработан подход MANUAL, предназначенный для разработки тестов, проверяющих базовую функциональность операционной системы. Эти тесты проверяли

работоспособность ключевых компонентов операционной системы перед началом тщательного тестирования тестовой системой, разработанной при поддержке инструментов CTESK.

Тесты MANUAL представляют собой код на языке программирования Си, использующий макросы и функции библиотеки поддержки MANUAL. Каждое тестовое испытание записывается в виде отдельной функции, начинающейся с макроса `TC_START("имя тестового испытания")` и заканчивающейся макросом `TC_END()`. Тело теста состоит из трех частей:

- подготовке окружения и данных;
- непосредственно тестового воздействия и проверки его результатов;
- освобождения ресурсов.

Проверка корректности работы тестируемой системы проводится при помощи функции `tc_assert(проверяющее_выражение, "текст, описывающий ошибку")`. Если проверяющее выражение оказывается равным false, то система считает, что произошла ошибка в тестируемой системе и выводит сообщение, описывающее ее. Кроме того, система автоматически отлавливает в ходе выполнения теста исключительные ситуации, что приравнивается к обнаружению ошибки.

Система MANUAL поддерживает иерархическую композицию тестовых испытаний в пакеты. Для запуска тестов предусмотрено два режима: автоматический и интерактивный. В автоматическом режиме система выполняет указанный набор тестов или пакетов и сохраняет журнал их выполнения. В интерактивном режиме пользователю предоставляется возможность навигации по дереву пакетов вплоть до индивидуального теста и возможность запустить выбранный тест или пакет на исполнение.

Основным недостатком системы MANUAL является ее слабая масштабируемость. Проблемы с масштабируемостью связаны с тем, что каждый отдельный тест представляет собой функцию на языке Си, что при росте тестового набора требует либо многочисленного дублирования кода, либо существенного объема достаточно рутинной ручной работы по структуризации кода.

Отсутствие параметризации отдельных тестов, которое вполне оправдано при реализации простейших проверок работоспособности базовой функциональности целевой системы, является существенным препятствием для применения подхода к разработке более детальных тестовых наборов.

2.2. Check

Система Check [3] предназначена в первую очередь для модульного тестирования ПО в процессе его разработки. Тем не менее, Check можно использовать и для тестирования программных интерфейсов на соответствие стандартам.

Check предоставляет разработчику тестов набор макросов и функций для выполнения проверок в тестах, для объединения тестов в наборы, управления выводом результатов и т.д.

Тест представляет собой код на языке программирования Си, заключённый между макросами START_TEST и END_TEST. Проверки требований в тестах выполняются с использованием функций fail_unless(`проверяющее_выражение, "текст, описывающий ошибку"`) и fail_if(`проверяющее_выражение, "текст, описывающий ошибку"`).

Как для каждого теста, так и для набора тестов можно задать функции, выполняющие инициализацию и освобождение используемых ресурсов (т.н. *checked u unchecked fixtures*)

К достоинствам системы Check стоит отнести:

- возможность выполнения каждого теста в отдельном процессе, т.е. изоляция тестов друг от друга и от самой среды Check;
- автоматическая обработка исключительных ситуаций в тестах;
- возможность задания максимально допустимого времени выполнения теста;
- специальные средства для проверки ситуаций, когда выполнение тестируемой функции приводит к посылке сигнала;
- интеграция системы сборки и запуска тестов с autoconf и automake — широко используемыми средствами автоматизации процесса сборки и установки ПО [4].

Система Check имеет, тем не менее, некоторые недостатки, из-за которых использование её не всегда оправдано.

- С помощью Check сложно разрабатывать параметризуемые тесты. Часто возникает ситуация, когда некоторую функцию необходимо проверить при разных наборах значений аргументов, а код теста при этом почти не меняется. Логично было бы передавать эти наборы значений аргументов в данный тест как параметры. Но в Check в качестве параметра тесту явно можно передать только его номер, что не очень удобно.
- Проверки, выполняемые в тесте, никак не привязаны к тем местам в документации по тестируемым функциям, где сформулированы соответствующие требования.
- Для добавления нового теста в набор необходимо перекомпилировать и код всех остальных тестов из этого набора, что не всегда удобно.
- Не поддерживается вывод стандартных кодов результата теста, определённых в стандарте на тестирование соответствия POSIX [5].

2.3. CUnit

Система CUnit [6] может использоваться в тех же случаях, что и Check [3], но обладает в целом более скромными возможностями.

Основным недостатком CUnit по сравнению с Check является то, что все тесты, а также система их запуска и сбора результатов, в данном случае выполняются в одном и том же процессе. Т.е. сбой какого-либо из тестов может привести к повреждению области памяти, используемой другими тестами или самой средой CUnit.

Также в отличие от Check в CUnit нет защиты от "зависания" тестов (нельзя задать максимально допустимое время работы теста).

CUnit имеет и некоторые преимущества по сравнению с Check.

- Поддержка т.н. fatal и non-fatal assertions. В первом случае, если проверка выявила нарушение требования, выполнение теста прекращается и остальные проверки в нём, таким образом, не выполняются (в Check всё происходит именно так). Если же нарушение требования выявлено в non-fatal assertion, выполнение теста продолжается. Возможно, остальные проверки в данном тесте позволят в таком случае дать разработчику более полную информацию о том, что происходит в тестируемой системе. Это может быть полезно при выяснении причины обнаруженного сбоя.
- Набор специальных функций и макросов, облегчающих выполнение часто используемых проверок: равенство и неравенство целых чисел, чисел с плавающей точкой, строк и т.д.
- Поддержка вывода отчётов о выполнении тестов в разных форматах, в том числе пригодных для отображения в web-браузере (xml+xslt).

Тем не менее, указанные в предыдущем разделе недостатки системы Check в полной мере относятся и к CUnit. Test Environment Toolkit (TET), о котором речь пойдёт ниже, свободен от некоторых из этих недостатков.

2.4. TET (Test Environment Toolkit)

Система TETware (TET, Test Environment Toolkit) достаточно широко используется для тестирования различных программных интерфейсов. Средства TET позволяют запускать разные тесты единым образом и получать отчёт о работе тестов в едином формате [7]. Информация о выполнении теста, включая его результат и сообщения, которые в нём выводятся, попадает в так называемый *журнал TET*.

Основными компонентами TET являются:

- test case controller (tcc) — этот компонент управляет запуском тестов и сбором выводимой ими информации;
- интерфейс прикладного программирования (TET API), который нужно использовать в тестах для того, чтобы их можно было

выполнять в среде TET. TET API существует для различных языков программирования, в том числе и для C/C++.

Наиболее существенными достоинствами TET, на наш взгляд, являются следующие:

- единая среда для запуска тестов;
- обработка исключительных ситуаций в тестах (например, segmentation fault) средствами test case controller;
- общие для всех тестов допустимые коды результата, соответствующие стандарту [5]: PASS, FAIL, UNRESOLVED и т.д., плюс возможность определить дополнительные коды результата тестов;
- возможность добавлять новые тесты в набор без перекомпиляции существующих тестов (использование т.н. *сценариев TET*)
- единый формат отчёта о выполнении тестов (*журнала TET*).

Эти преимущества TET облегчают анализ результатов работы тестов. В частности, программные средства для обработки журнала TET при построении статистики по результатам тестов могут не учитывать специфику проведённых тестов.

С другой стороны, средства TET, в основном, автоматизируют выполнение тестов и сбор результатов их работы. TET не предоставляет ни инструментов для автоматизации разработки тестов, ни API для выполнения проверок в тестах. Соответственно, есть несколько причин, из-за которых применение TET "в чистом виде" (без использования каких-либо "надстроек" над ним) не очень удобно.

- Отсутствие средств для связи проверок, проводимых в тестах, с текстом соответствующего стандарта.
- Нередко приходится писать тесты, код которых почти одинаков, например, отличаются только параметры вызываемых функций или, допустим, типы элементов используемых массивов и т.п. Возникает естественное желание автоматизировать разработку таких тестов, так, чтобы можно было многократно использовать общие части их кода. К сожалению, в составе TET нет инструментов, позволяющих сделать это.
- Разработчику тестов необходимо добавить в код определения нужных для запуска в среде TET специфических функций, структур и т.д., что вполне можно было бы сделать и автоматически.
- Тесты, запускаемые с помощью test case controller, не всегда удобно отлаживать. Как для поиска ошибок в тесте, так и для исследования поведения тестируемой системы в случае сбоя полезно исключить влияние средств TET на работу теста, что существенно упрощает использование программ-отладчиков (например, gdb).

Ниже речь пойдёт как раз о "надстройках" над TET (системы GTKVTS и T2C), в которых перечисленные выше недостатки в той или иной степени преодолены.

2.5. Автоматизация разработки TET-совместимых тестов в GTK+-2.0 Verification Test Suite (GTKVTS)

Подход, использующийся в GTK+-2.0 Verification Test Suite (GTKVTS) при разработке TET-совместимых тестов, позволяет преодолеть некоторые из описанных выше недостатков TET [8].

Во-первых, в GTKVTS используются т.н. параметризуемые тесты. То есть разработчик пишет шаблон кода тестов на обычном C, отмечая в нём особым образом места, куда впоследствии будут вставлены значения параметров теста. Для каждого такого шаблона может быть задано несколько наборов параметров. Параметром может быть практически что угодно, не только параметр тестируемой функции или её ожидаемое возвращаемое значение. Иногда удобно вынести в качестве параметров типы используемых данных (наподобие template в C++) или даже сам вызов тестируемой функции и т.д.

По шаблону кода теста для каждого набора параметров генератор C-кода GTKVTS создаёт отдельную функцию на языке C (см. Рис. 1).

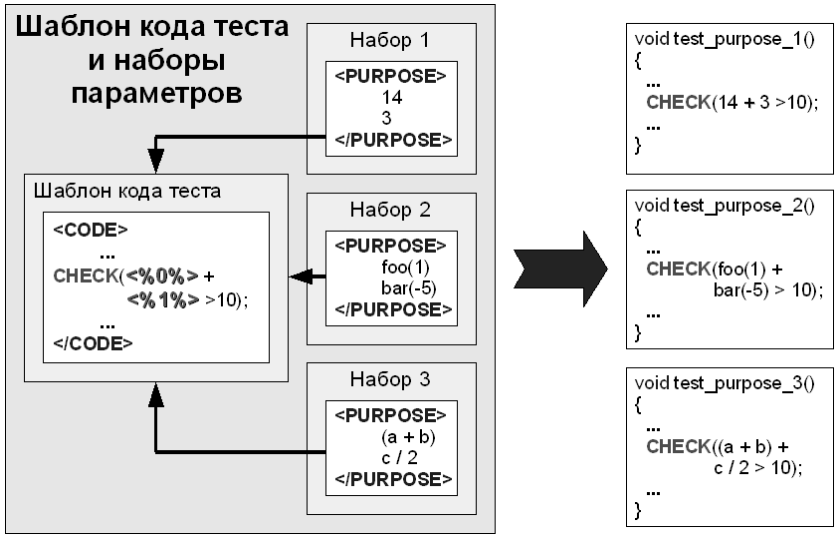

Рис. 1. Генерация C-кода тестов по шаблону. "<%0%>" и "<%1%>" в шаблоне кода теста — места для вставки параметров.

Во-вторых, при генерации кода тестов на C средства GTKVTS автоматически вставляют в этот код определения данных, необходимых для работы теста в среде TET, так что разработчику не приходится заботиться об этом самому.

Помимо этого, автоматически создаются и make-файлы, необходимые для сборки тестов, а также файлы сценариев TET, что тоже удобно.

В GTKVTS также сделана попытка связать выполняемые в тестах проверки с соответствующими фрагментами текста стандарта: в комментариях перед каждым тестом разработчик должен указать текст проверяемых в данном тесте требований. К сожалению, в самом тесте при выполнении проверок этот текст никак не используется, а по трассе теста сложно понять, какие именно требования проверялись и какие из них нарушаются в тестируемой системе.

К менее существенным недостаткам средств GTKVTS стоит также отнести отсутствие поддержки отладки теста вне TET, а также специализированность инструментальных средств для использования при разработке тестов только для библиотек из стека GTK+.

2.6. Достоинства и недостатки существующих решений

Мы рассмотрели 5 подходов к разработке тестов для программных интерфейсов на языке программирования Си. Достоинства и недостатки каждого из них сведены в Таблице 1.

	MANUAL	Check	CUnit	TET	GTKVTS
Параметризация тестов	–	–	–	–	+
Прослеживаемость требований стандарта	–	–	–	–	–
Запуск тестов в отдельном процессе	–	+	–	–	–
Автоматическая обработка исключительных ситуаций	+	+	–	+	+
Ограничение времени работы теста	–	+	–	+	+
Иерархическая организация пакетов	+	–	–	–	–
Удобство отладки тестов	+	+	+	–	–
Переносимость тестов	–	+	+	+	–
Использование стандартных видов вердиктов [5]	–	–	–	+	+

Таблица 1. Сравнение существующих подходов.

Все рассмотренные подходы обладают определёнными достоинствами. Тем не менее, с точки зрения тестирования программных интерфейсов на соответствие требованиям стандарта все они обладают существенным недостатком, заключающемся в отсутствии поддержки прослеживаемости проверок в тестах к требованиям стандарта. Кроме того, ни один из этих

инструментов не сочетает в себе все указанные выше достоинства, хотя явных противоречий между ними не существует.

При разработке набора инструментов T2C была поставлена задача обеспечить прослеживаемость требований стандарта, реализовав при этом все достоинства существующих решений, представленные в Таблице 1, за исключением иерархической организации пакетов тестов. Данное исключение связано с тем, что возможность иерархической организации пакетов не оказывает существенного влияния на разработку и эксплуатацию тестов на соответствие требованиям стандарта.

Система T2C ("Template-to-Code")

2.7. Общие сведения

Система T2C ("Template-to-Code"), позволяет автоматизировать разработку параметризованных тестов, работающих как в среде TET, так и вне её.

Код тестов на языке C создается на основе T2C-файла, в котором расположены шаблоны тестов и наборы параметров для них (схема та же, что и для GTKVTS — см. Рис. 1). Фрагмент T2C-файла показан ниже.

Тесты, которые будут созданы по шаблону, показанному на Рис. 2, имеют два параметра: *TYPE* и *INDEX*. В первом из сгенерированных тестов в качестве *TYPE* будет использоваться int, в качестве *INDEX* — 6, во втором — double и 999, соответственно.

Также, как и в GTKVTS, в код теста на C автоматически добавляются все необходимые определения данных и функций для выполнения этого теста в среде TET, а также генерируются make-файл и файл сценария TET.

Таким образом, инструменты T2C сохраняют основные достоинства системы GTKVTS, но при этом они поддерживают рекомендации по разработке тестов соответствия, сформулированные, в частности, в [9]:

- составление каталога элементарных требований к тестируемым программным интерфейсам;
- привязка проверок, выполняемых в тестах, к соответствующим местам в тексте стандарта;
- оценка качества тестирования в терминах покрытия элементарных требований.

В T2C также сделаны следующие усовершенствования по сравнению с GTKVTS.

- Разработчику тестов предоставляется набор высокоуровневых программных интерфейсов (T2C API), с помощью которых и выполняются проверки в тестах. При этом, если проверка показывает, что некоторое требование стандарта нарушено, в трассу теста (журнал TET) выводится, помимо всего прочего, текст этого требования.

- Есть возможность создания независимой версии теста на чистом C/C++, не использующей средства TET. Это может быть полезно для отладки самого теста или для более тщательного исследования поведения тестируемой системы в случае отладки ошибки в реализации.
- Заготовки для T2C-файлов создаются автоматически по размеченному тексту стандарта.
- В T2C-файле можно указать код, необходимый для инициализации и освобождения ресурсов, общих для всех тестов, код которых будет создан по этому файлу, а также для освобождения ресурсов, выделенных в данном тесте.
- Поддерживается выполнение тестов в отдельных процессах.
- Есть возможность задать максимальное допустимое время работы теста. Это полезно в тех случаях, когда какие-то из тестов могут "зависнуть".

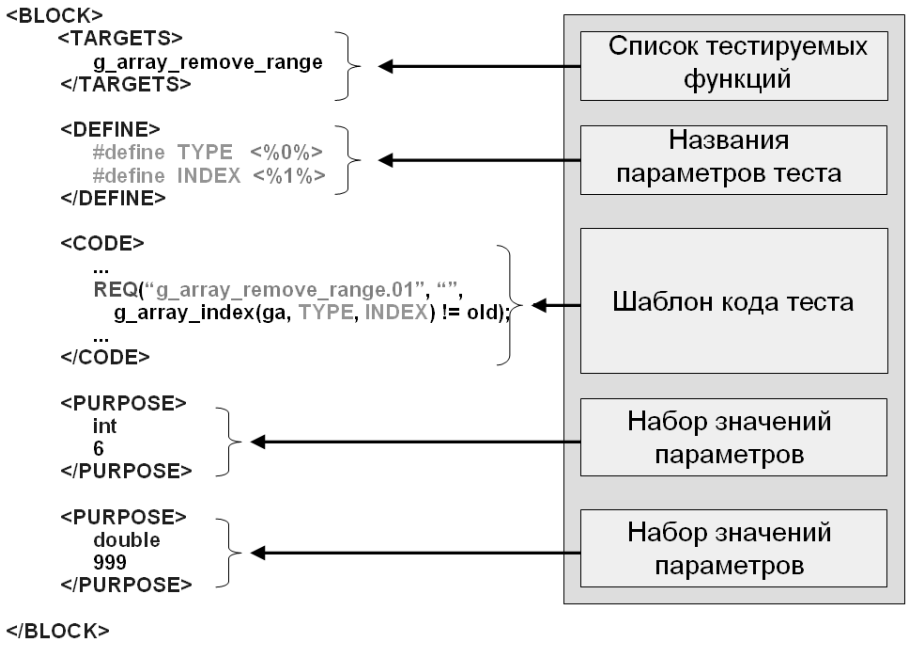

Рис. 2. Фрагмент t2c-файла.

2.8. Процесс разработки тестов с помощью системы T2C

В этом разделе рассматриваются основные стадии процесса разработки тестов с помощью T2C.

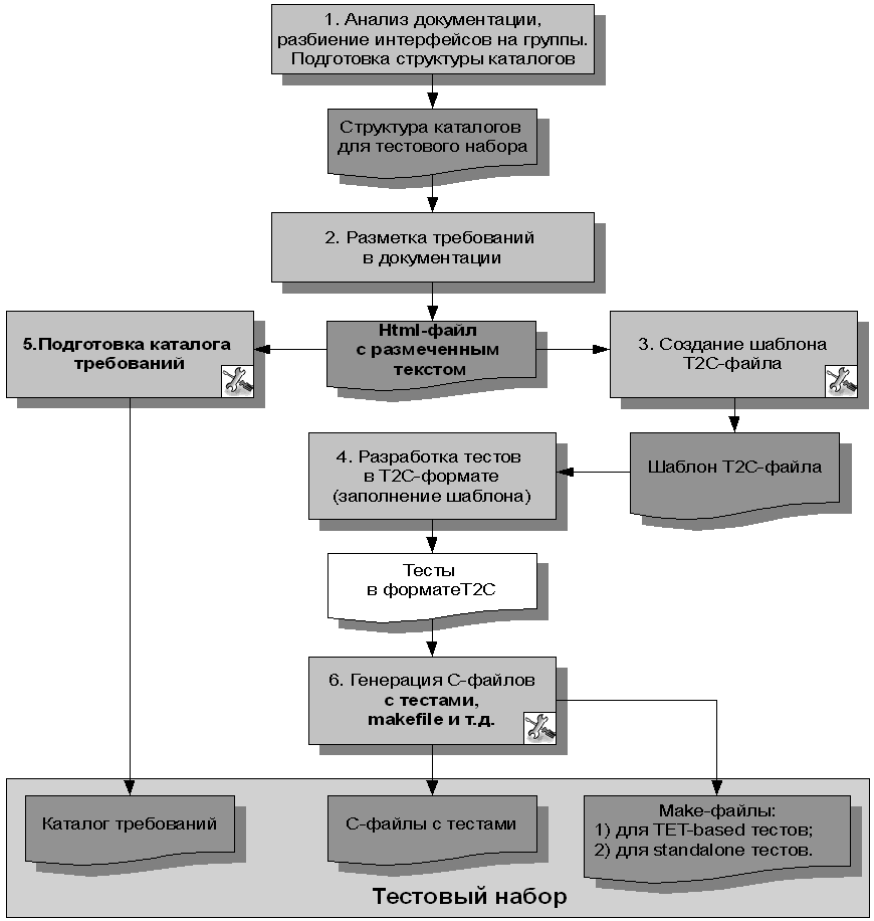

Рис. 3. Разработка тестов с помощью системы T2C.

3.8.1. Анализ документации и разбиение тестируемых интерфейсов на группы

Прежде, чем приступать к разработке тестов для набора программных интерфейсов, необходимо изучить описание этих интерфейсов в документации, выяснить, что именно предстоит тестировать. При этом стоит разбить интерфейсы на группы, каждая из которых реализует определённую часть функциональности тестируемой системы. Стоит избегать ситуаций, когда, например, для проверки интерфейсов из группы A требуются интерфейсы из группы B ("A зависит от B"), и при этом для тестирования

группы B требуются интерфейсы из A (циклическая зависимость групп интерфейсов).

Иногда разбиение интерфейсов на группы уже проведено в документации. Например, документация по библиотеке Glib [10] состоит из таких разделов, как "Arrays", "Unicode Manipulation", "Memory Allocation" и т.д. Интерфейсы, описанные в каждом таком разделе, как правило, принадлежат к одной функциональной группе.

При разработке тестов для группы интерфейсов создаётся один или более T2C-файлов. На этом этапе удобно создать и необходимую структуру каталогов для тестового набора.

Ниже предполагается, что документация по тестируемым интерфейсам является набором html-документов.

3.8.2. Разметка требований в документации

На данном этапе в документации необходимо выделить элементарные требования к тестируемым интерфейсам и присвоить каждому требованию уникальный идентификатор [9]. При необходимости текст требований можно переформулировать для улучшения читаемости.

Разметка требований выполняется в html-редакторе KompoZer (www.kompozer.net) с использованием инструмента ReqMarkup, разработанного в рамках проекта OLVER [11] и затем интегрированного в систему T2C.

3.8.3. Создание шаблона T2C-файла

Когда выделены и размечены все требования для интерфейсов из данной функциональной группы, с помощью ReqMarkup автоматически создаётся заготовка T2C-файла.

3.8.4. Разработка тестов в T2C-формате

Этот этап самый важный в процессе создания тестов. На данном шаге разработчику нужно заполнить заготовку T2C-файла, добавляя туда шаблоны кода тестов и наборы параметров для них. Также, если необходимо, в специальных секциях файла следует указывать код инициализации и освобождения ресурсов, общих для всех тестов, код которых будет сгенерирован по этому файлу.

При редактировании T2C-файла может оказаться полезным компонент T2C Editor — plug-in для Eclipse IDE. Он обеспечивает удобную навигацию по секциям T2C файла, предоставляет средства для работы с параметрами тестов и т.д.

3.8.5. Подготовка каталога требований

По размеченной документации с помощью инструмента ReqTools, входящего в состав T2C, в web-браузере создаётся каталог требований для данной группы интерфейсов. Этот каталог используется при работе теста: если проверка в тесте показывает, что некоторое требование нарушено, из каталога загружается формулировка этого требования с соответствующим идентификатором и выводится в трассу теста.

3.8.6. Генерация кода тестов, make-файлов и сценариев TET

Когда подготовлены тесты в T2C-формате и создан каталог требований, следует запустить генератор кода (T2C Code Generator), который создаст C-файлы с кодом тестов, make-файлы для их сборки, сценарии TET и т.д.

3.8.7. Сборка, запуск и отладка тестов

На данном этапе нужно провести сборку тестового набора, используя make-файлы, сгенерированные на предыдущем шаге. После чего можно запускать готовые тесты в среде TET или отлаживать какие-либо из них вне TET.

3. Применение T2C при разработке тестов для LSB Desktop

Система T2C использовалась (и используется в настоящее время) при разработке тестов для интерфейсных операций ("*интерфейсов*") библиотек Linux, входящих в стандарт Linux Standard Base (LSB). С её помощью подготовлены тесты базовой функциональности для следующих библиотек:

- Glib (libglib-2.0);
- GModule (libgmodule-2.0);
- ATK (libatk-1.0);
- Fontconfig (libfontconfig).

В Таблице 2 представлены результаты тестирования. Описания найденных ошибок опубликованы на http://linuxtesting.ru/results/impl_reports.

Библиотека	Версия	Всего интерфейсов	Протестировано	Найдено ошибок в реализации
libatk-1.0	1.19.6	222	222 (100%)	11
libglib-2.0	2.14.0	847	832 (98%)	13
libgmodule-2.0	2.14.0	8	8 (100%)	2
libfontconfig	2.4.2	160	133 (83%)	11
Всего		**1237**	**1195 (97%)**	**37**

Таблица 2. Результаты тестирования LSB библиотек тестами T2C.

Примечание 1. В столбце "*Версия*" указывается последняя версия соответствующей библиотеки на момент публикации тестового набора. Количество найденных ошибок указано именно для этой версии. По обнаруженным ошибкам идёт активная работа совместно с разработчиками соответствующих библиотек, так что в более поздних версиях некоторые или даже все из этих ошибок могут быть исправлены.

Примечание 2. В столбце "*Всего интерфейсов*" указано общее количество входящих в LSB интерфейсных операций ("*интерфейсов*") библиотеки, включая и недокументированные. Практически все документированные интерфейсы были протестированы.

Средние затраты на полный цикл разработки теста (от разметки и анализа требований до отлаженного кода теста) для одного интерфейса составили порядка 0,5-1 человеко-дня.

Отметим, что интерфейсы в документации по данным библиотекам далеко не всегда описаны подробно. В среднем для каждого интерфейса было выделено 2-3 элементарных требования.

Сведения о количестве выделенных элементарных требований к тестируемым интерфейсам и о покрытии этих требований представлены в Таблице 3.

Библиотека	Выделено требований	Проверено требований	Покрытие по требованиям, %
libatk-1.0	515	497	96%
libglib-2.0	2461	2290	93%
libgmodule-2.0	21	17	80%
libfontconfig	236	177	75%
Всего	**3233**	**2981**	**92%**

Таблица 3. Выделенные и проверяемые требования для LSB библиотек.

4. Заключение

Задача тестирования программных интерфейсов на соответствие документации (в том числе стандартам) является важным звеном в обеспечении качества и интероперабельности программных систем. Для решения этой задачи разрабатываются различные методики и инструменты, позволяющие систематизировать и автоматизировать работу. При этом всегда возникает выбор баланса между качеством тестирования и затратами на разработку соответствующих тестов. Решение здесь обычно принимается директивно из достаточно субъективных соображений. Тем временем, с выбором целевого качества тестирования связан и выбор оптимальной технологии и инструментов, так как для различных уровней цены и качества необходимы разные подходы. Так, например, для глубокого тестирования важных интерфейсов хорошо зарекомендовала себя технология UniTESK [12],

которая, однако, требует и соответствующих высоких затрат как на освоение технологии, так и на разработку самих тестов.

В данной работе рассмотрена технология T2C, нацеленная на эффективную разработку тестов «среднего уровня» для проверки базовой функциональности программных интерфейсов. В данном случае понятие «средний уровень» соответствует наиболее распространенным представлениям о качестве производственного тестирования, наблюдаемом в большинстве изученных авторами тестовых наборов (например, сертификационные тесты Open Group, сертификационные тесты LSB, тесты OpenPosix и Linux Test Project). T2C позволяет повысить эффективность разработки таких тестов за счет использования следующих основных возможностей, минимизирующих ручной труд на создание окружения и дублирование кода, не принципиального для специфической логики конкретного теста:

- Автоматическая генерация заготовок тестов на основе каталога требований.
- Использование именованных параметров в коде тестов, для каждого набора значений которых создается соответствующий экземпляр теста.
- Высокоуровневое API для использования в коде тестов для осуществления проверок и трассировки.
- Генерация независимых отдельных тестов в виде самодостаточных простых программ на C/C++, что в практическом плане существенно упрощает отладку тестов и реализации (в отличие от отладки в полной среде запуска TET или подобной).

Среда выполнения тестов, созданных по технологии T2C, опирается на широко распространенные средства TETware, что позволяет легко интегрировать тесты в существующие тестовые наборы и окружения управления тестированием и анализа результатов. Кроме того, важной особенностью технологии T2C является систематизированная работа с каталогом требований к проверяемым интерфейсам и обеспечение связи отдельных требований с проверками в тестах и соответствующими сообщениями в отчете о результатах тестирования.

Технология T2C была успешно применена в Институте системного программирования РАН в проекте [13] по разработке сертификационных тестов для проверки соответствия библиотек Linux стандарту LSB. В работе представлена статистика по разработанным тестам, найденным ошибкам и трудозатратам, которая позволяет сделать вывод об эффективности данной технологии для заданного сегмента качества разработки тестов программных интерфейсов различных модулей и библиотек. В настоящее время реализована инструментальная поддержка Си и Си++, но нет принципиальных проблем и для других современных языков программирования общего назначения, например C# или Java. Стоит отметить, что важным фактором для успешного применения T2C является наличие достаточно устоявшегося текста

требований, так как этап анализа и составления каталога требований в условиях частых изменений и плохого качества описаний может составлять существенную часть трудозатрат.

В дальнейшем в T2C предполагается реализовать средства для автоматизации работы с множествами значений параметров тестов. Будет повышена и степень интеграции инструментов T2C со средой разработки Eclipse. Также предполагается исследовать возможности интеграции систем T2C и CTESK.

Литература

[1] Домашняя страница CTESK. http://www.unitesk.com/products/ctesk.
[2] The Linux Standard Base. http://www.linux-foundation.org/en/LSB.
[3] Домашняя страница Check. http://check.sourceforge.net/doc/check.html/index.html.
[4] Домашняя страница Autoconf и Automake. http://www.gnu.org/software/automake/.
[5] IEEE.2003.1-1992 IEEE Standard for Information Technology — Test Methods for Measuring Conformance to POSIX — Part 1: System Interfaces. IEEE, New York, NY, USA, 1992. ISBN 1-55937-275-3.
[6] Домашняя страница CUnit. http://cunit.sourceforge.net/.
[7] TETware User Guide, http://tetworks.opengroup.org/documents/3.7/uguide.pdf.
[8] GTKVTS Readme. http://svn.gnome.org/viewvc/gtkvts/trunk/README.
[9] В.В.Кулямин, Н.В.Пакулин, О.Л.Петренко, А.А.Сортов, А.В.Хорошилов. Формализация требований на практике. Препринт ИСП РАН, Москва, 2006.
[10] Glib Reference Manual, http://www.gtk.org/api/2.6/glib/.
[11] Центр верификации ОС Linux. http://linuxtesting.ru/.
[12] Домашний сайт UniTESK. http://unitesk.com/.
[13] Домашняя страница проекта LSB Infrastructure. http://ispras.linux-foundation.org/.