Унификация в автоматизации тестирования. Позиция UniTESK

А. К. Петренко
petrenko@ispras.ru

Аннотация. Рассматриваются современные тенденции унификации в средствах автоматизации тестирования, анализируется текущий уровень унификации в инструментах семейства UniTESK и очерчиваются возможные формы унификации, которые позволят расширить круг возможностей технологии.

1. Введение. Специализация и унификация в развитии UniTESK

Первые работы по созданию новой технологии тестирования, которая вскоре была названа UniTESK, начались в ИСП РАН во второй половине 1999 года. Само название включало в себя один из корней слова унификация (UNI-fication), а в целом расшифровывалось как **Uni**fied **TE**sting **S**pecification tool**K**it. Унификации в UniTESK постоянно придавалось большое значение. Одним из важных элементов унификации был принцип использования однородного спецификационного расширения языков программирования — во всех расширениях использовалась одна и та парадигма спецификации (контрактные спецификации) и единообразная архитектура теста, включающая одинаковые механизмы взаимодействия тестовой системы с целевой системой. Этот принцип позволил, следуя одной общей методике, создавать тесты для программного обеспечения на различных языках программирования.

В основу новой технологии был положен опыт разработки и использования технологии KVEST [1] и опыт ее использования в задачах тестирования программного интерфейса (API, Application Program Interface) ядра операционной системы компании Nortel Networks [2]. Кроме того, рассматривались сценарии применения новой технологии при тестировании реализаций телекоммуникационных протоколов и других распределенных и компонентных систем. Предшественник UniTESK, технология KVEST, была в некотором роде «еще более унифицированной». Она в качестве языка формального описания функциональных требований и тестовых сценариев использовала язык формальных спецификаций RSL [3], что, в принципе,

позволяло разрабатывать спецификации и тесты независимо от языков программирования, которые используются в реализации.

Опыт KVEST показал, что в чем-то принятые принципы унификации оказались достаточно удачными, а в чем-то совсем неприемлемыми. К удачным можно отнести подход к описанию функциональных требований в форме контрактных спецификаций — в виде ограничений типа пред- и постусловия и ограничений на типы данных. К неудачным приходится отнести принцип использования единого, универсального языка спецификаций. Причем первым основанием для критики такой декларации является совсем не ограниченность возможностей языка (хотя RSL достаточно консервативный язык, он не имеет объектно-ориентированных возможностей, в нем практически нет средств для описания темпоральных свойств и др.). Главным тезисом критики RSL является его непохожесть на распространенные языки программирования, что сразу вызывает сложности как в обучении будущих пользователей, так и в совместном использовании программ на языке RSL и языке программирования во время разработки и отладки теста.

Выбирая принципы унификации UniTESK, мы сделали небольшой шаг назад, сказав, что языки спецификаций для разных языковых платформ реализации могут быть разными. Так появилось семейство однородных спецификационных расширений для разных языков: для Java — J@va, для языка C — SEC (Specification Extension of C) и Chase — спецификационное расширение для C#.

Одновременно с разработкой новых инструментов семейства UniTESK мы расширяли круг использования этих инструментов. Поскольку о тестировании реализаций протоколов мы думали заранее, опыт применения CTESK (UniTESK для C-платформы) не потребовал кардинальных переделок в инструменте. Однако при анализе возможностей UniTESK-инструментов для системного тестирования компиляторов стало ясно, что для этого требуются новые техники, так появились инструменты BNF tool и OTK.

Затем были попытки применения UniTESK для тестирования графических пользовательских интерфейсов и их разновидности — Web-интерфейсов. Часть попыток была вполне успешной, тесты удавалось построить достаточно быстро, их было легко модернизировать и синхронизовать с эволюционирующей реализацией. Однако в общем случае, оказалось, что помимо чисто технических проблем управления графическим интерфейсом имеются и принципиальные ограничения UniTESK в плане возможностей по описанию/спецификации пользовательского интерфейса как такого.

В последние 3 года спектр работ, в которых проводились эксперименты по применению UniTESK, резко расширился. В него вошли тестирование распределенных платформ интеграции крупных информационных систем, тестирование ОС реального времени, тестирование библиотек ОС Linux, тестирование системы поддержки времени исполнения Java, реализаций

протоколов Mobile IPv6, IPsec, IPMP и др. Расширение спектра приложений, естественно, повлекло необходимость развития имевшихся инструментов UniTESK, необходимость разработки новых и использования инструментов, имеющихся на рынке. В итоге иногда складывается впечатление, что для каждой новой задачи приходится искать (а, если найти не удается найти, то создавать) новый инструмент. С одной стороны, специализация — это тоже известный путь развития. Не бывает универсальных самолетов, автомобилей, даже в более прозаических вещах, таких как ложки и ножи, также нет универсальных решений. Однако универсальность и унификация — это не одно и тоже. Унификация — это средство, позволяющее на основе некоторого типового набора возможностей получить решение широкого круга задач. Интуиция подсказывает, что в тестировании может быть построен набор средств, достаточно ограниченный, из которых как из Lego-конструктора можно собрать нужный инструмент. Анализу достигнутого уровня унификации в автоматизации тестирования и направлениям развития унификации и посвящается данная статья.

2. Унификация в программировании и в автоматизации тестирования

Начнем с определения термина «Унификация».

> ***Унификация*** (от лат. unus — один и facio — делаю) — приведение к единообразию, к единой форме или системе (Большой энциклопедический словарь, М., БСЭ, 1964).
>
> ***Унификация*** — в технике, приведение различных видов продукции и средств её производства к рациональному минимуму типоразмеров, марок, форм, свойств и т.п. Основная цель У. — устранение неоправданного многообразия изделий одинакового назначения и разнотипности их составных частей и деталей, приведение к возможному единообразию способов их изготовления, сборки, испытаний и т.п. У.— важное направление в развитии современной техники, комплексный процесс, охватывающий вопросы проектирования, технологии, контроля и эксплуатации машин, механизмов, аппаратов, приборов. В условиях научно-технической революции принципы У. используют не только в отраслях производства, но и в др. сферах человеческой деятельности... (Большая советская энциклопедия**.** БСЭ**,** 1969-1978 [4]).

В качестве ключевых понятий, которые определяют цели и формы унификации, авторы этой статьи называют:

- взаимозаменяемость;
- типизация;
- построение агрегатных унифицированных систем;

- качество выпускаемой продукции, её надёжность и долговечность благодаря более высокой технологичности конструкции изделий и проработанной технологии их изготовления;
- стандартизация;
- единые системы технической документации;
- специализация производства.

Похоже, авторы БСЭ затронули практически все моменты, которые являются ключевыми в автоматизации тестирования. Пожалуй, по понятным причинам, осталась не затронутой только одна тема — интероперабельность компонентов тестовых систем.

2.1. Унификация в использовании и в разработке средств тестирования

Каков опыт унификации в программировании, каковы удачные примеры унификации в программировании? Самый первый общепризнанный результат унификации — это библиотеки программ (раньше они назывались библиотеками стандартных программ). Собственно, с создания библиотек и средств их использования началось программирование как новая научная и техническая дисциплина. Следующим направлением унификации можно назвать стандартизацию языков программирования. Второй, возможно, не такой заметный, но чрезвычайно важный пример унификации — это определение стандартных представлений программ, позволяющих объединять объектные модули, оттранслированные с различных языков программирования в единую программную систему. Это как раз тот случай, когда мы не можем предложить совершенно общего, универсального решения при выборе языка программирования, но имеем средства унификации, чтобы, комбинируя модули на разных языках, решить любую стоящую перед нами задачу. Хороший пример унификации — общая схема построения компиляторов, которая предусматривает фазы лексического, синтаксического, семантического анализа, вместе они образуют так называемый front-end компилятора, фазы оптимизации и генерации кода, которые вместе образуют так называемый back-end. Такая разбивка на фазы позволяет не только упростить изучение устройства компиляторов, становится реальной возможность построения компилятора методом сборки отдельных компонентов. Можно собрать несколько фронт-ендов (для разных языков) и несколько бэк-ендов для разных аппаратных платформ и получить компилятор, который объединяет в себе наилучшие качества, реализованные в его отдельных модулях. Унификация в компиляторах проводится и на уровне инструментов разработки компиляторов. Например, имеются средства типа Lex и Yacc, при помощи которых можно сгенерировать модули фронт-енда.

Рассматривая вопросы унификации в автоматизации тестирования, следует обратить внимание на то, что имеется два ракурса этого рассмотрения. Первый ракурс — это позиция разработчика тестов, второй — позиция

разработчика средств автоматизации тестирования. Начнем с анализа позиции тестировщика, или разработчика тестов.

Переходя от проекта к проекту или переключаясь с разработки тестов для одной системы к другой, тестировщик отвечает на однотипные вопросы:

- На каких входных данных нужно проверять систему?
- Что является критерием правильности функционирования системы?
- Что является критерием полноты тестирования? и др.

В зависимости от особенностей интерфейса системы, от возможностей взаимодействия с системой (например, за счет прямого доступа к ее внутреннему состоянию), от требований заказчика и других условий тестировщик выбирает (или получает в свое распоряжение) набор инструментальных средств для разработки тестов. По крайней мере, из соображений экономии времени, очевидно, что разработка тестов будет выполняться быстрее, если в новом инструментарии тестировщик увидит привычные средства с привычными возможностями. Это возможно только при условии, что при разработке различных инструментов следуют некоторой общей схеме унификации[1].

Теперь вернемся к унификации с точки зрения разработки самих инструментов автоматизации тестирования. По сути, все инструменты автоматизации тестирования выполняют ограниченный набор операций. Сложность этого набора можно сравнить со сложностью компиляторов, недаром, некоторые генераторы тестов называются трансляторами (например, изначальное название генератора тестов ADLT расшифровывалось как Assertion Definition Language Translator [5]). Если аналогия верна, то, казалось бы, что в распоряжении разработчиков инструментов генерации тестов уже давно должны были появиться генераторы входных тестовых данных, генераторы тестовых оракулов, выносящих вердикт на основе того или иного представления требований, анализаторы тестового покрытия в соответствии с тем или иным критерием покрытия и так далее. У разработчиков компиляторов есть еще один «инструмент унификации» — это компилятор GCC, который можно рассматривать как коллекцию типовых решений задач разработки компиляторов. На основе такой «коллекции» многие строят свои собственные компиляторы методом инкрементальных модификаций. Какова же ситуация с унификацией в нашем случае? Ответ пугает своей категоричностью — инструментов, которые позволяли бы воспользоваться типовыми решениями перечисленных выше задач тестирования, нет. Одной из причин этого служит то, что нет единого подхода ни к форме представления

[1] Заметим, что временной фактор в оценке эффективности процесса разработки тестов не является единственным и, возможно, не является главным. Унификация ведет к постоянному росту профессионального мастерства, что, в свою очередь, ведет к повышению качества тестирования.

входных данных для программ таких библиотек, ни к форме представления их результатов.

2.2. Обзор работ по унификации и стандартизации средств тестирования

Тема унификации технологий и инструментов тестирования широко и активно обсуждается. Сейчас трудно сказать, является ли этот интерес некоторой модной темой или в программной индустрии имеется реальная заинтересованность, подпитывающая исследования и разработки (одним из оснований для сомнений в реальной заинтересованности является то, что активность в этой области заметна в Европе и мало заметна в США и Азии).

Большая часть исследований и публикаций по этой теме так или иначе опираются на UML 2.0 Testing Profile (U2TP) [6], который развивается Консорциумом U2TP [7]. Это профиль унифицированного языка моделирования систем UML 2.0. U2TP претендует стать стандартом в данной области и является развитием подхода Model Driven Architecture (MDA), который продвигается группой OMG [8]. В Консорциум U2TP входят такие известные компании как Ericsson, IBM, Motorola, Telelogic и один из ведущих немецких исследовательских институтов Fraunhofer FOKUS.

U2TP ограничивает спектр рассмотрения процессов тестирования тестированием по методу «черного ящика». Сам он рассматривается как систематическая модель для описания компонентов тестовых систем и их возможных взаимодействий в ходе выполнения тестов с помощью конструкций UML 2.0 и базируется на метамодели UML. Такие описания в дальнейшем используются как артефакты, позволяющие автоматизировать часть работ по созданию и исполнению тестов. Одна из задач U2TP состоит в систематизации терминов, которые используются при описании архитектуры тестов, целей тестирования (test purposes), тестовых вариантов (test cases) и средств их привязки к тестируемым системам (адаптеров), процессов тестирования и участников процесса.

Помимо инициативы U2TP в Европе имеется еще несколько проектов, например, два проекта из программы ITEA (Information Technology for European Advancement) [9], которая, в свою очередь является ветвью программы Eureka [10].

- EAST/EEA (Electronics Architecture and Software Technology — Embedded Electronic Architecture) [11]. Суть этого проекта, объединяющего 23 партнера из 4 стран, — разработка профиля UML 2.0, который станет языком описания архитектуры программно-аппаратных систем в автомобильной промышленности. В частности, отдельным разделом в разрабатываемом профиле выделены процессы верификации и валидации, включающие тестирование, и набор модельных компонентов, используемых в рамках этих процессов.

- Проект TT-Medal [12], цель которого — продвижение наукоемких методов тестирования, в особенности с использованием языка TTCN3, в промышленную практику европейских компаний. Практическим результатом проекта стала интеграция трех инструментов: TTworkbench [13], включающего компилятор TTCN-3, среду разработки тестов в Eclipse, генераторы кодеков; Classification Tree Editor (CTE) [14], средство для ручного построения классов эквивалентности в системах с большим числом параметров и генератор тестов финской компании Conformiq [15]. В рамках данного проекта рассматривались различные способы построения интерфейса между тестирующей системой и тестируемой. В проекте была продемонстрирована возможность использования описаний интерфейсов на ASN.1, IDL, и XML.

Стоит упомянуть еще два проекта из программы ITS Framework Programme 6.

- HIDENETS (Highly Dependable IP-based Networks and Services) [16] нацелен на разработку устойчивой к отказам архитектуры (resilience solutions) для распределенных и мобильных систем (mobility-aware services). Так же, как и в других проектах, которые мы здесь рассматриваем, большое внимание уделяется не только самим целевым системам, но инфраструктуре для моделирования, верификации и валидации. Этим объясняется использование в проекте U2TP. В контексте данной статьи интересным является полный обзор стандартов, которые затрагивают вопросы унификации и переиспользования артефактов как ПО в целом, так и артефактов из области тестирования. В частности, достаточно подробно анализируется RAS: Reusable Asset Specification [17], также разработанный под патронажем OMG.
- ATESST (Advancing Traffic Efficiency and Safety through Software Technology) [18]. Одним из результатов этого проекта является отчет по теме Linking Requirements and Test Artifacts [19]. В нем подробно перечисляются различные способы выделения требований и их привязки к различным артефактам и процессам проектов, включая тесты и их отдельные компоненты.

В качестве объективной оценки достижений в данной области можно сослаться на то, что многие работы, которые изначально поддерживались европейскими программами, теперь нашли свое продолжение в рамках альянса AUTOSAR [20], который финансируется индустриальными партнерами. Сейчас список партнеров альянса включает практически всех крупных поставщиков автомобилей и электроники для них.

Вместе с тем, имеется по крайней мере два аспекта, в которых результаты активности, связанной с U2TP, можно было оценить как недостаточно значимые (естественно, здесь мы рассматриваем только проблемы тестирования): спектр рассматриваемых проблем и подходов к их решению, а

также доведение результатов до реальных техник, технологий, программных продуктов и автоматизированных сервисов.

Начнем с последнего аспекта. Сейчас основной формой практических результатов являются стандарты, описывающие архитектуры и процессы тестирования. Пока эти стандарты являются некоторыми рекомендациями, они не обрели четкой и строгой формы, необходимой для того, чтобы быть переведенной на язык интерфейсов компьютерных систем. Причем, наряду с организационными трудностями, которых всегда много при создании новых стандартов, похоже, есть трудности, обусловленные нестыковкой подходов при описании артефактов и ролей на абстрактном уровне и реальными технологиями, методами и инструментами, которые используются в производстве.

Возможно, корни этой проблемы лежат в сфере первого из указанных выше аспектов, а именно в том, что идеологически подход U2TP и язык разработки тестов TTCN-3 следуют логике разработки ручных тестов и практически не уделяют внимания собственно методам генерации тестов. По различным соображениям приверженцы U2TP пытаются сохранить представление о тестовом варианте (test case) как о простом сценарии. Сложные модели, из которых могут извлекаться тесты, в концепции U2TP просто не рассматриваются. Профиль ограничивается описанием компонентов самих тестов, без всякой оглядки на способы их получения.

Идея «черного ящика» как единственного рассматриваемого подхода к тестированию также позволяет разработчикам U2TP закрывать глаза на проблемы анализа структуры реализации, без чего нельзя решить, например, такие важные задачи как построение модели тестового покрытия и модели ошибок, учитывающей особенности реализации и хорошо зарекомендовавшие себя на практике эвристики.

Эти ограничения в постановке задачи унификации не позволяют решить ее с использованием наиболее перспективных подходов к автоматической генерации тестов. В частности, на основе U2TP невозможно обеспечить интеграцию различных передовых техник верификации и валидации программ, таких как проверка моделей (model checking), верификационный мониторинг[2] (run-time verification), генерация тестов методами статического анализа или с помощью генетических алгоритмов и др. Сложность интеграции этих методов обусловливается тем, что при эффективном решении задачи автоматической генерации тестов для реальных больших систем часто нельзя выделять решение отдельных задач генерации в модули, достаточно слабо связанные друг с другом. Простейший пример: пусть нам надо отобрать представителей различных классов эквивалентности из некоторого

[2] Русский перевод термина «run-time verification» предложен В. В. Кулямины.

дискретного многомерного пространства, являющегося областью определения некоторой функции. Логически задачу можно разбить на следующие три шага.

- Построить все наборы значений — точки пространства.
- Разбить полученное множество на классы эквивалентности.
- Выбрать из каждого класса по одному произвольному представителю.

На практике такой последовательный подход и, соответственно, простая схема композиции модулей, которые отвечают за выполнение каждого шага, не работают по простой причине — число точек в области определения, хотя и конечно, но не может уместиться ни в каких разумных вычислительных ресурсах. Вместе с тем эта задача имеет решение, которое требует как можно более прямой генерации наборов значений в заданных классах эквивалентности.

В заключение данного критического анализа хотелось бы отметить, что хотя концептуально важно отделять друг от друга методы тестирования на основе «белого ящика» и на основе «черного ящика», и хотя тестирование на основе моделей, конечно, является одним из методов «черного ящика», но технически автоматическая генерация тестов на основе моделей во многом не отличается от методов генерации на основе исходных кодов (или другого представления реализации). Поэтому декларация U2TP об использовании в качестве базового подхода исключительно метода «черного ящика», будучи понятой буквально, является барьером на пути более полной автоматизации генерации тестов.

3. Специализация и унификация в UniTESK. Текущее состояние

3.1. Задачи построения тестов

В работах по методам тестирования и по автоматизации тестирования затрагиваются многочисленные задачи, которые необходимо уметь решать при создании тестов. Примерный список этих задач представлен ниже.

- Определение требований к системе, выделение из них условий корректного воздействия на целевую систему и критериев правильности результатов[3].
- Построение оракула — компонента теста, выносящего вердикт о выполнении или нарушении требований на основе результатов

[3] Некоторые теоретики программной инженерии рассмотрение этого вопроса отнесли бы к дисциплине, которая называется управлением требованиями (requirements management), однако на практике, даже если в начале проекта требования сформулированы, тестировщики должны к ним отнестись критически и проанализировать, насколько имеющиеся формулировки требований пригодны как основа для разработки тестов. Часто проектирование тестов влечет переформулировку и пополнение требований.

работы этого теста. Более детальное рассмотрение этой задачи вскрывает новые вопросы: что является источником критериев правильности, удается ли, и как именно, описать такие критерии в общем виде, как они трансформируются в оракул, как выносится общий вердикт теста, если в нем несколько оракулов, и т.п.

- Определение критериев полноты тестирования.
- Сбор информации о достигнутом покрытии.
- Использование информации о достигнутом покрытии для управления ходом генерации.
- Собственно генерация тестовых воздействий, которая распадается на следующие подзадачи.
  - Построение последовательности тестовых воздействий в целом.
  - Выбор вызываемой операции в рамках одного воздействия, если интерфейс предоставляет несколько операций.
  - Генерация входных данных операции.
    - Простейшая итерация данных из области определения операции, без анализа других моделей (требований, ошибок, структуры системы).
    - Итерация простых структур данных.
    - Итерация иерархических структур данных.
    - Нацеливание итерации входных данных при использовании моделей требований, ошибок или структуры тестируемой системы.
    - Статическое вычисление входных данных для достижения покрытия ситуаций в различных моделях, в том числе и в структуре системы.
  - Отбрасывание некорректных тестовых данных.
  - «Прореживание» тестовых данных, то есть обеспечение покрытия того же множества существенно различных в соответствии с используемыми критериями покрытия ситуаций при помощи меньшего количества отдельных воздействий.
- Генерация отчетов о тестовом покрытии.
  - Отчеты на основе каталогов неформальных требований.
  - Отчеты на основе структурных характеристик реализации (для различных метрик тестового покрытия).
  - Отчеты на основе формальных моделей, при этом рассматриваются модели реализации, модели требований, модели ошибок. Примерами возможных видов моделей

являются контрактные спецификации, исполнимые модели (конечные автоматы, LTS, сети Петри, MSC диаграммы и пр.), грамматики и производные от них структуры, например графы зависимости и деревья вывода, другие формализмы описания структурированных данных (SQL, XML scheme, Relax NG, ASN 1.0 и др.).

- Связь теста с целевыми интерфейсами, в том числе конвертация данных, их транспорт, marshalling, перехват событий; эти задачи часто связываются с элементами тестовых систем, которые называют адаптерами, медиаторами, брокерами, proxy и др.
- Служебные задачи, например, сбор трассировочной информации для целей анализа обнаруженных ошибок или вычисления достигнутого покрытия, то есть для генерации отчетов различных видов; конфигурационное управление системой тестов и ассоциированной с ними информации (каталоги требований, об ошибках, запросы на их исправление, трассировка процесса устранения ошибок и т.п.).

Попробуем рассмотреть материал данного сборника сквозь призму унификации систем автоматизации разработки тестов. Заметим, что статьи сборника представляют достаточно широкий спектр походов и техник. Так в работах Д. Силакова и В. Мутилина за основу взят метод «белого ящика», используется инструментация исходного кода, все остальные работы — классический пример использования подхода «черного ящика». В работах А. Демакова, С. Зеленова, С. Зеленовой, Р. Зыбина, В. Кулямина, А. Пономаренко, В. Рубанова и Е. Чернова тесты строятся, в основном, на основе изучения области входных данных. В. Рубанов, А. Хорошилов, Е. Шатохин представляют технологию традиционной ручной разработки тестов, в статьях А. Камкина и М. Чупилко описывается подход к тестированию модулей микропроцессоров. Часть работ использует прием тестирования на лету (testing-on-the-fly, генерацию тестовых данных и воздействий в ходе выполнения теста). Повышению эффективности локализации ошибок по результатам тестирования на лету посвящена статья С. Грошева. В статье А. Камкина рассматриваются методы статической генерации тестов (тестовых программ), в статье В. Кулямина — методы определения критериев тестового покрытия, в статьях И. Бурдонова и А. Косачева разрабатывается формальный подход к определению понятия конформности — соответствия поведения системы некоторой модели — и методы проверки этого соответствия.

Возьмем, например, первые два пункта из списка задач, приведенного выше.

- Определение требований к системе.
- Построение оракула.

В работах, описывающих разработку тестов на соответствие стандарту LSB, подробно рассматриваются выделение требований для тестов проверки базовой работоспособности (инструмент Azov) и для тестов традиционного

типа (инструмент T2C). Кроме того, в этих же работах упоминаются тесты более высокого качества для LSB Core, созданные в рамках проекта OLVER [21] при помощи инструмента CTESK. Все эти работы используют общую базу данных, в которой хранятся разнообразные данные об интерфейсах. Данные о сигнатурах интерфейсных функций используются всеми тремя инструментами — здесь унификация уже достаточно высокого уровня. Вместе с тем, в работе по модульному тестированию микропроцессоров, где используется тот же инструмент CTESK, возможности единообразного хранения требований к интерфейсам не использованы.

Рассматривая те же работы в плане унификации механизма вынесения вердикта можно обнаружить, что три инструмента, использующиеся для разработки тестов для LSB строят оракул совершенно по-разному, то есть в этом аспекте унификации пока нет.

3.2. Вектор развития UniTESK

Если рассмотреть все задачи, которые необходимо решить при автоматизации создания тестов, то для каждой из них в отдельности можно предложить некоторую систематизацию подходов к решению этих задач. Разработка такой системы — это первый шаг к унификации. Но, похоже, что этого шага пока не сделали разработчики U2TP — они предложили номенклатуру компонентов, из которых состоят тестовые системы, но не сделали попытки более подробного описания каждой сущности, попавшей в номенклатуру.

После первого шага должен следовать второй. Он состоит в разработке шаблонов проектирования. Это нелегкая задача, но здесь она еще более усложняется из-за того, что эти шаблоны должны учесть варианты совместного и согласованного решения нескольких задач автоматизации тестирования. Примером таких взаимосвязанных задач является целенаправленная генерация тестовых данных для покрытия заданных операторов или заданного пути в коде и «прореживание» комбинаций входных данных, если их общее количество слишком велико. При этом стратегия генерации и стратегия «прореживания» часто должны динамически адаптироваться в ходе тестирования.

Шаблоны согласованных решений должны появиться и для тех случаев, когда можно совместно использовать разные техники тестирования или верификации, например, элементы проверки моделей (model checking) и статического анализа с применением инструментов, автоматически находящих данные, удовлетворяющие сложным ограничениям (constraint solvers). Другое место, где потребуются специальные усилия для согласования разных подходов — использование нескольких разных моделей, служащих для описания требований, поведения, гипотез о возможных ошибках и т.п.

На первый взгляд может показаться, что задача унификации, если ее решать достаточно основательно, становится неоправданно сложной. Но судить об этом нужно, соотнося трудоемкость ее решения и эффект, который может

быть получен. Результат, к которому должна привести унификация средств разработки инструментов автоматизации тестирования, с одной стороны, должен обеспечить совместимость компонентов тестов друг с другом, а, с другой стороны, выведет нас на возможность интеграции различных методов и инструментов верификации.

Сколько шагов унификации уже пройдено технологией UniTESK? Прямого ответа на этот вопрос пока дать нельзя, но что-то уже сделано. Есть задел в плане введения единой номенклатуры артефактов, ролей и процессов тестирования. Наработан некоторый опыт унифицированных решений для разработки тестов совершенно разного плана (например, база данных интерфейсов для систем, тестирующихся при помощи инструментов Azov, T2C, CTESK). Начаты работы по интеграции JavaTESK и генератора сложных структур данных Pinery, ведется работа по интеграции Pinery и алгоритмов «прореживания» на основе покрывающих наборов [22]. То есть, сейчас есть некоторый опыт в унификации как в рамках программы первого шага, так и второго, однако полной картины унифицированной схемы разработки инструментов тестирования пока нет. Создание такой схемы — это и есть новый вектор развития технологии.

4. Заключение

В статье рассматривались проблемы и перспективы унификации в области разработки систем автоматического построения тестов. Были перечислены положительные и отрицательные моменты, сопровождающие любую унификацию и унификацию разработке программных систем, в частности. Были выделены два ракурса унификации в автоматизации тестирования: позиция разработчика тестов и позиция разработчика инструментов создания тестов. Далее в статье был проанализирован опыт в унификации систем тестирования сконцентрированной вокруг работ по использованию MDA подхода, реализованного в U2TP — UML 2.0 Testing Profile. Была показана ограниченность постановки задачи, определенной авторами U2TP.

На примере работ, представленных в сборнике, показано, какие решения в рамках разрабатываемых в ИСП РАН технологий уже в некоторой степени унифицированы, какие пока далеки от унификации. Отмечено, что унификация в автоматизации тестирования может проводиться на основе компонентного подхода. Это подразумевает, что должны выделяться модули, играющие те или иные роли, должны описываться интерфейсы этих модулей, типовые шаблоны проектирования (patterns), которые на основе стандартизованного набора модулей позволят строить инструменты, с учетом как специфики тестируемой системы, так и методов спецификации и тестирования, выбранных в данном инструменте.

Заключая статью, хотелось бы, во-первых, отметить, что задача унификации в автоматизации тестирования расценивается как в высшей степени актуальная, во-вторых, что опыт европейских исследователей и специалистов показывает

на трудности, которые встают при попытках проведения унификации методом «сверху-вниз». Вероятно, коренная причина здесь в идеологии подхода MDA — реализацию можно построить на основе модели, если адекватная модель, или хотя бы схема моделирования уже есть. Такое бывает лишь в областях, где накоплен большой опыт разработки типовых решений. Поскольку область автоматической генерации тестов бурно развивается и проблем в ней еще больше, чем решений, здесь пока таких моделей нет, и не понятно, когда они появятся. Констатация этого факта не означает, что унификация невозможна. Она возможна, но подходить к решению этого вопроса надо с двух сторон: от общего к частному, как в MDA, и от частного к общему, как это всегда происходит в программировании, если разработчик постоянно думает об обобщении своего опыта, о выделении тех модулей и тех операций, которые повторяются многократно в разных проектах, при решении разных задач. Замечу, что объектами такого переиспользования могут быть не только программные модели (что каждый раз является настоящей удачей), такими объектами могут быть шаблоны проектирования, протоколы взаимодействия отдельных компонентов и другие артефакты и процессы, которые составляют как сами системы автоматизации тестирования, так и процессы их разработки и эксплуатации.

5. Благодарности

Автор выражает признательность А. В. Бойченко за предоставление большого набора материалов по стандартизации профилей процессов тестирования и В. В. Кулямину за многочисленные обсуждения проблем унификации в тестировании.

Литература

[1] I. Burdonov, A. Kossatchev, A. Petrenko, D. Galter. KVEST: Automated Generation of Test Suites from Formal Specifications. Proceedings of Formal Method Congress, Toulouse, France, 1999, LNCS 1708:608–621, Springer-Verlag, 1999.
[2] http://www.nortel.com.
[3] The RAISE Language Group. The RAISE Specification Language. Prentice Hall Europe, 1992.
[4] http://slovari.yandex.ru/dict/bse.
[5] http://adl.opengroup.org/exgr/icse/icse98.htm.
[6] http://www.omg.org/technology/documents/formal/test_profile.htm.
[7] http://www.fokus.fraunhofer.de/u2tp/index.html.
[8] http://www.omg.org.
[9] http://www.itea2.org.
[10] http://www.eureka.be.
[11] http://www.east-eea.net.
[12] http://www.tt-medal.org/.
[13] http://www.testingtech.de/products.
[14] http://www.systematic-testing.com/functional_testing/cte_main.php?cte=1.
[15] http://www.conformiq.com.
[16] http://www.hidenets.aau.dk/.

[17] http://www.omg.org/technology/documents/formal/ras.htm.
[18] http://www.atesst.org.
[19] http://www.atesst.org/scripts/home/publigen/content/templates/show.asp?P=114&L=EN&ITEMID=5.
[20] http://www.autosar.org.
[21] http://linuxtesting.org/.
[22] C. J. Colbourn. Combinatorial aspects of covering arrays. Le Matematiche (Catania) 58:121–167, 2004.