УДК 004.054; 004.582

А.С. ПРИГОЖЕВ

Одесский национальный политехнический университет, Украина

ЯЗЫКОНЕЗАВИСИМАЯ СРЕДА РАЗРАБОТЧИКА ДЛЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В работе предлагаются подходы к проектированию среды поддержки разработчика в процессе отладки программного обеспечения. Рассматриваются существующие решения в области тестирования программного обеспечения. На основе анализа сформулирована задача построения среды поддержки разработчика, независимой от синтаксических конструкций конкретного языка программирования. Задачами данной среды является визуализация процесса отладки программного обеспечения, автоматизированный поиск ошибок, а также конвертация существующих исходных текстов на одном языке программирования в исходный текст на другом языке программирования. Базой для построения такой среды является граф, ориентированный на функциональную структуру программы. Предлагается структурная схема среды, описаны основные алгоритмы ее функционирования.

Ключевые слова: *тестирование программ, поддержка разработчика, автоматизация тестирования.*

Введение

Разработка современных программных систем является весьма сложным и трудоемким процессом. Значительную часть времени при разработке системы занимает процесс тестирования и отладки программного обеспечения. В средах разработки программного обеспечения существует большое количество средств, направленных на облегчение процесса отладки. К ним относятся окна просмотра состояний переменных, встроенный механизм точек останова, окно изменения значений переменных и т.д.

Однако перед разработчиком практически всегда стоит задача интерпретации информации, выдаваемой отладчиком, а также семантически верной корректировки кода. Современный отладчик, как правило, позволяет вывести значения переменных процесса, значение стека, и другие параметры. Недостаток данного подхода - отсутствие возможности запомнить состояние переменных для каждого состояния алгоритма, что зачастую необходимо для анализа его работы.

Другая, не менее важная проблема, возникает при модификации кода программного обеспечения. Некоторые модули системы могут быть реализованы на языке программирования, однако плохо документированы разработчиком. Поэтому часто возникает проблема понимания разработчиком исходного кода, который необходимо модифицировать. Особенно данная проблема актуальна для программного обеспечения с открытым исходным кодом.

Таким образом, существует необходимость в дальнейшем развитии средств отладки программного обеспечения в направлении визуализации данных и алгоритмов работы и усовершенствования средств, упрощающих работу с исходным кодом. В настоящее время существует ряд технологий, позволяющих в значительной степени автоматизировать процесс тестирования.

1. Существующие средства автоматизации тестирования

Большинство современных систем автоматизации тестирования ориентированы на конкретные классы приложений, как например система автоматизации тестирования Web-приложений, основанная на скриптовых языках, предложенная в [1]. Данный подход основан на анализе исходного кода (технология тестирования «белого ящика»). В процессе тестирования используются некоторые особенности кода приложений, предназначенных для работы в Web.

При анализе кода в данной методологии в первую очередь используется извлечение переменных и их значений. Основными переменными, извлекаемыми из скрипта, являются массивы $GET и $POST, в которых хранится основная информация о взаимодействии пользователя с сайтом. Весь анализ запросов к Web-серверу осуществляется на основе анализа указанных переменных.

Кроме выделения параметров и их значений в данной разработке используется исследование зависимостей между переменными. При анализе запросов выделяются зависимости двух типов: зави-

симости от переменных и зависимости от значений переменных. В первом случае предполагается, что присутствие некоторых переменных в запросе зависят от присутствия других переменных. Во втором случае на присутствие переменных в запросе влияет не только формальное наличие некоторой переменной, но и на ее значение.

Третьей важной составляющей анализа является анализ на соответствие спецификации HTML, а также исследование файлов протокола Web-сервера на наличие в нем ошибок.

Достоинством данного метода является возможность анализировать исходный код программы с учетом значений переменных (т.н. алгоритм «белого ящика»). Недостатком метода является достаточно ограниченная сфера применения.

Еще одной важной технологией автоматизации тестирования программного обеспечения является технология UniTESK [2], позволяющая разрабатывать тесты для программного обеспечения. Основная идея архитектуры теста UniTesK состоит в том, что разрабатывается набор компонентов для тестирования различных видов ПО с использованием разных стратегий тестирования. Эти компоненты должны иметь четко определенные обязанности в системе и интерфейсы для взаимодействия друг с другом. Далее, информация, которую в общем случае может предоставить только разработчик тестов, концентрируется в небольшом числе компонентов с четко определенными ролями. Для каждого такого компонента разрабатывается компактное и простое представление, создание которого потребует минимальных усилий со стороны пользователя.

Архитектура теста UniTesK [3] основана на следующем разделении задачи тестирования на подзадачи:

1. Задача проверки корректности поведения системы в ответ на единичное воздействие.

2. Задача создания единичного тестового воздействия.

3. Задача построения последовательности таких воздействий, нацеленной на достижение нужного покрытия.

4. Задача установления связи между тестовой системой, построенной на основе абстрактного моделирования, и конкретной реализацией целевой системы.

Для построения тестов в технологии UniTesK используется конечно-автоматная модель программной системы, а для тестирования параллелизма и распределенных систем используются специальные типы конечных автоматов [4]. Для построения тестов строится конечно-автоматная модель целевого программного обеспечения в виде тестовых сценариев. Проход по данным сценариям обеспечивает генерацию некоторого теста. Тестовая последовательность строится во время тестирования динамически, за счет построения некоторого "исчерпывающего" пути по переходам автомата. Это может быть обход всех его состояний, всех его переходов, всех пар смежных переходов и т.п. Алгоритм построения такого пути на достаточно широком классе автоматов оформлен в виде другого компонента теста, обходчика.

Достоинством данного подхода является использование графовой модели для тестирования ПО. К недостаткам можно отнести отсутствие формального метода построения графовой модели.

Таким образом, при построении систем тестирования часто используется анализ исходных данных и тестирование на основе моделей. Однако данный анализ чаще всего строится на выборке конкретных значений данных в определенный момент алгоритма. Для тестирования на основе моделей основной проблемой является отсутствие формальных подходов к построению моделей анализа.

Современные средства тестирования программного обеспечения можно улучшить, разработав методы визуализации сложных структур данных в программе, обеспечив независимость среды тестирования от языка программирования, на котором реализована программа и реализовав в такой системе экспертный анализ информации получаемой в процессе тестирования.

2. Модификация графа сценариев

В качестве средства для решения поставленной задачи является разработка методов и алгоритмов представления исходного кода в виде некоторого графа. Данный подход был предложен в [5]. Однако остается проблема формального построения такого графа. Одно из решений данной проблемы, позволяющее формально построить дерево, используя код программы, представлено в работе [6]. Структура дерева базируется на алгоритмической алгебре, для которой доказана ее функциональная полнота [6]. Таким образом, на данном дереве может быть отображена любая схема алгоритма. При его обходе в прямом порядке, с учетом условий альтернативы и цикла, приходим к выполнению алгоритма. Данный подход схож с построением тестовых сценариев в работах [2-3].

Для реализации указанного похода структуру дерева необходимо расширить, введя в него информацию о номере шага алгоритма, на котором выполняется тот или иной оператор, а также идентификатор оператора. Данная информация используется в дальнейшем для связи с данными, которые обрабатывает алгоритм.

В работе [6] предлагается модель дерева сценариев, которая позволяет отобразить любой алгоритм на граф задач. Каждой вершине дерева поставим в соответствие номер с помощью алгоритма обхода дерева в прямом порядке. Это позволяет пронумеровать операции алгоритма в том же порядке, в каком они записаны в алгоритме. Обход дерева в прямом порядке с учетом условий цикла и альтернативы будет означать выполнение алгоритма. При таком обходе дерева также будут проставляться номера вершин, причем, если при обходе некоторые вершины будут посещены несколько раз, то этим вершинам будет присвоено несколько номеров, идущих последовательно, что позволяет сопоставить одинаковым операторам, которые выполняются на различных шагах алгоритма разные данные.

Любой алгоритм всегда связан с данными, обрабатываемыми им. Большая часть ошибок, выявляемых в процессе выполнения алгоритма, либо связана с доступом к данным либо отражается непосредственно на данных. Рассмотрим структуры данных, которые позволяли бы контролировать изменение данных в ходе выполнения программы.

3. Представление данных алгоритма для анализа программы

Любой алгоритм оперирует данными одного из следующих типов: целый, вещественный, логический, символьный, массив, строка, указатель, структура. Для проведения дальнейшего синтеза системы необходимо несколько изменить классификацию переменных. В дальнейшем будем рассматривать переменные трех типов: данные, указатели и записи. К классу «данные» отнесем целые, вещественные, логические и символьные переменные. К классу «указатели» относятся переменные-указатели и индексы массивов. К классу «запись» относятся массивы и собственно записи. В переменных данного класса присутствует хотя бы по одной переменной, принадлежащей классам «данные» и «указатель».

Для целей отладки необходимо для каждого из перечисленных классов переменных сопоставить шаг алгоритма, после которого изменилось значение данной переменной, идентификатор оператора, изменившего значение данной переменной, а также дополнительные параметры. Это позволит анализировать изменения переменных во времени, а также визуализировать их.

Рассмотрим далее дополнительные характеристики, которые необходимо сопоставить каждому введенному классу переменных. Для переменных относящихся к классу данных необходимо знать информацию, связанную с типом значения данных (целый, вещественный и т.д.), минимальным и максимальным значением для данного типа в данном языке программирования и т.п.

Значение переменной класса «указатель» – это индекс переменной или структуры данных, на которую они указывают, в соответствующей таблице. Если указатель является указателем на массив, то элементы массива в таблице данных должны храниться последовательно. Также для всех указателей предполагается, что они являются некоторыми положительными числами в разрешенном диапазоне значений. Интервал разрешенных значений устанавливается в зависимости от архитектуры операционной системы разработчиком системы.

Если переменная принадлежит классу «запись» то она должна храниться в системе как список соответствующих идентификаторов данных и указателей.

Для каждого из классов необходимо также хранить текущее значение, номер оператора и номер шага алгоритма на котором возникло указанное изменение переменной.

Зная указанные характеристики можно построить некоторую последовательность наборов значений переменных на каждом шаге алгоритма. Анализ данных, представленных в таком виде позволяет провести визуализацию, что облегчает процесс тестирования, и дает возможность осуществлять экспертный анализ разработанной программы.

4. Структура системы языконезависимой отладки ПО

Система языконезависимой отладки программного обеспечения должна обеспечивать возможность преобразования исходного текста программы из текста на любом языке программирования или спецификации в представление в виде некоторого дерева, аналогичного представленному в работах [6-8]. Далее построенным деревьям должна быть сопоставлена информация о данных программы, для которой строится дерево. Для решения изложенных задач предлагается реализовать специальный блок преобразования исходного текста, который учитывает особенности построения синтаксических конструкций конкретного языка программирования и может их преобразовать в выражения алгебры Дейкстры [9-11]. С точки зрения процедурно-ориентированных языков программирования минимальной структурной единицей для преобразования в выражение в алгебре Дейкстры является функция в описании процедурно-ориентированного языка. Каждая функция преобразовывается в выражение на языке алгебры Дейкстры. Если функция содержит вызовы других функций, то она добавляется в данное выражение.

После построения выражения в алгебре Дейкстры оно преобразовывается согласно соотношениям в алгебру сценариев, а на основе уже данного выражения строится граф задач. Построенный граф задач, совместно с добавленными в него переменными рассматривается системой тестирования с целью анализа программы на соответствие алгоритму и выявлению ошибок (рис. 1).

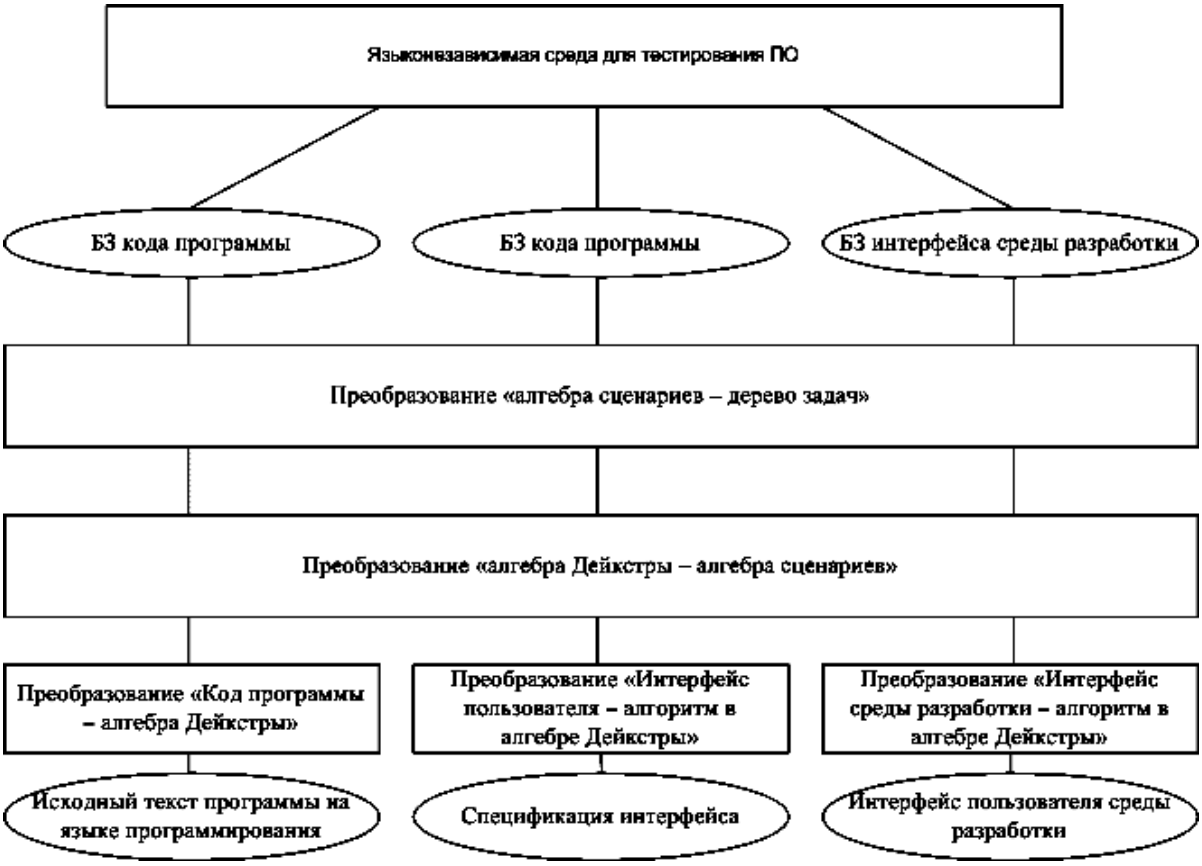

Рис. 1. Структура среды тестирования программ

Аналогично представленному алгоритму для кода преобразовывается интерфейс пользователя приложения (см. рис. 1), при этом переменными в этом случае являются те элементы, с которыми взаимодействует пользователь, а функциями – элементы группировки команд в интерфейсе: пакетные командные файлы, элементы группировки графического интерфейса и т.п.

Вся информация о дереве и сопоставленным ему переменным хранится на диске в виде реляционных таблиц. Структура таблиц, представляющих дерево, представлена на рис. 2.

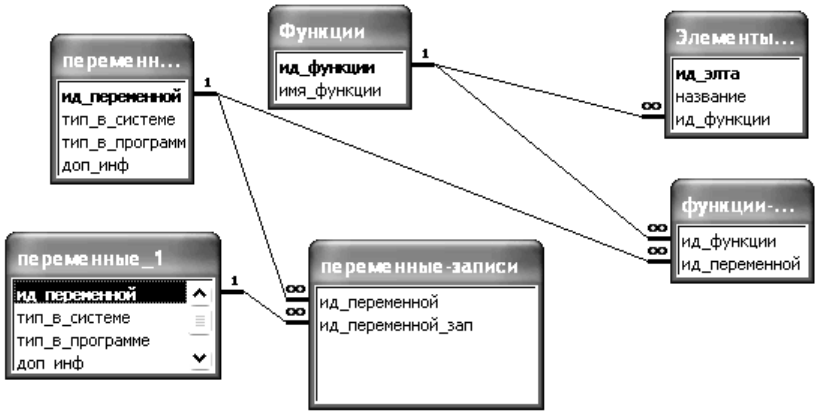

Рис. 2. Структура базы данных для хранения дерева

Таблица «Функции» представляет собой описание набора пользовательских функций в программе. Она состоит из уникального идентификатора функции и названия функции. Идентификатор функции в дальнейшем представлен в описании параметров, а также используется для связи элементов управления интерфейса пользователя с реализующими их функциями.

Таблица «Элементы интерфейса» указывает на тип элемента, а также функцию, реализующую взаимодействие с данным компонентом интерфейса. Одним из параметров данной функции является собственно значение элемента интерфейса.

Таблицы «Функции-параметры» связывает идентификаторы функций с идентификаторами переменных – формальных параметров этих функций. Таблица «Переменные» содержит все описания связанные с переменными, согласно классификации, приведенной в п. 4 настоящей статьи.

Таблица «Переменные-записи» включает в себя сведения о переменных, являющихся полями переменной типа «запись». Данная таблица содержит ссылку на строку в таблице «Переменные», которая описывает соответствующее поле записи.

Выводы

В данной статье были рассмотрены основные подходы к синтезу современных систем автоматизации тестирования программного обеспечения. Указано, что данные системы обладают рядом существенных недостатков, в частности отсутствием средств для формального построения графа, представляющего структуру программы, а также средств для представления и дальнейшего анализа состояния переменных во время представления программы.

Для решения указанных проблем было предложено использовать формальные методы трансформационного преобразования программ [10], а также метод построения дерева задач по известной схеме алгоритма [6]. Использование данных подходов позволяет скрыть синтаксическую структуру реализации программы, выделив при этом необходимую информацию, необходимую для отладки программы.

Предложена структура системы автоматизированной отладки и структура базы данных для хранения информации о коде и данных выполняемой программы. Дальнейшее развитие системы заключается в разработке методов анализа последовательности данных и деревьев с учетом действий пользователя в интерфейсе.

Синтез таких методов позволит значительно облегчить сопровождение приложений для разработчика, а также позволит создавать формальные отчеты о работе пользователя.

Литература

1. Силаков Д.В Автоматизация тестирования web-приложений, основанных на скриптовых языка: Труды института системного программирования РАН / Д.В Силаков [Электрон. ресурс]. – Режим доступа к ресурсу: http://www.citforum.ru/SE/testing/web_app/.

2. Баранцев А.В. Подход UniTesK к разработке тестов: достижения и перспективы / А.В. Баранцев, И.Б. Бурдонов, А.В. Демаков, С.В. Зеленов, А.С. Косачев, В.В. Кулямин, В.А. Омельченко, Н.В. Пакулин, А.К. Петренко, А.В. Хорошилов [Электрон. ресурс]. – Режим доступа к ресурсу: http://www.citforum.ru/SE/ testing/unitesk.

3. Bourdonov I. UniTesK Test Suite Architecture / I. Bourdonov, A. Kossatchev, V. Kuliamin, A. Petrenko // Proc. of FME 2002. – Springer-Verlag, 2002. – LNCS 2391. – P. 77-88.

4. Zafiropulo P. Towards Analysing and Synthesizing Protocols / P. Zafiropulo, C.H. West, H. Rudin, D.D. Cowan, D. Brand // IEEE Trans. on Communications. – April 1980. – COM-28(4). – P. 651-660.

5. Пригожев А.С. Принципы построения интеллектуальной среды разработчика программного обеспечения на основе экспертной системы / А.С. Пригожев // Искусственный интеллект 2008. Интеллектуальные системы 2008: Материалы IX межд. науч.-техн. конф. – пос. Кацивели, Крым, Украина. – 22-27 сент. 2008. – Донецк-Таганрог-Минск, 2008. – Т. 2. – С. 127-131.

6. Пригожев А.С. Информационная технология помощи пользователю / А.С. Пригожев // Холодильная техника и технология. – 2007. – № 2. – С. 98-101.

7. Рувинская В.М. Разработка планирующей экспертной системы для поддержки работы пользователя с программной системой / В.М. Рувинская, А.С. Пригожев // Вестник ХГТУ. – 2005. – №3. – С. 133-137.

8. Рувинская В.М. Построение подсистемы помощи пользователю ПК с использованием сценариев / В.М. Рувинская, А.С. Пригожев // Искусственный интеллект. – 2004. – № 3.– С. 371-378.

9. Цейтлин Г.Е. Введение в алгоритмику / Г.Е. Цейтлин. – К.: Сфера, 1998. – 310 с.

10. Цейтлин Г.Е. Инструментарий конструирования экспертных систем символьной обработки / Г.Е. Цейтлин, Т.К. Терзян, Л.М. Захария // Математические машины и системы. – 1997. – №1. – С. 14-24.

11. Цейтлин Г.Е. Критерий функциональной полноты в алгебре Дейкстры / Г.Е. Цейтлин // Кибернетика и системный анализ. – 1995. – №5. – С. 28-30.

Поступила в редакцию 5.02.2009

Рецензент: д-р техн. наук, проф. С.Г. Антощук, Одесский национальный политехнический университет, институт компьютерных систем, Одесса, Украина.

МОВНОНЕЗАЛЕЖНЕ СЕРЕДОВИЩЕ РОЗРОБНИКА ДЛЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

О.С. Пригожев

У роботі запропоновано підходи до проектування середовища підтримки розробника в процесі налагодження програмного забезпечення. Розглянуто існуючі рішення в області тестування програмного забезпечення. На основі аналізу сформульована задача побудови середовища підтримки розробника, незалежної від синтаксичних конструкцій конкретного язика програмування. Задачами даного середовища є візуалізація процесу налагодження програмного забезпечення, автоматизований пошук помилок, а також конвертація існуючих вихідних текстів на інший язик програмування. Базою для побудови такого середовища є граф, зорієнтований на функціональну структуру програми. Пропонується структурна схема середовища, описано її функціонування.

Ключові слова: тестування програм, підтримка розробника, автоматизація тестування.

SOFTWARE TESTING LANGUAGE INDEPENDENT ENVIRONMENT FOR PROGRAM ENGINEER

A.S. Prigozhev

In article the approaches to design a support medium for the program engineer during a debugging of software were offered. Existing solutions in the field of testing the software are considered. On the basis problem analysis construction of language independent support medium for the development engineer was formulated. Problems of the given medium is visualization of process of a debugging of the software, the automated navigation of errors, and also conversion of existing source texts in one programming language in the source text in other programming language. Baselines for construction of such medium are the graph oriented on the functional structure of program. The skeleton diagram of a medium was offered, the fundamental algorithms of its performance are presented.

Keywords: software testing, development engineer support, automation of testing.

Пригожев Александр Сергеевич – канд. техн. наук, ст. преп. кафедры системного программного обеспечения Одесского национального политехнического университета, Одесса, Украина, e-mail: prigozhev@rambler.ru.