Висновки. У роботі запропоновано структура викладання ІТ дисциплін для студентів некомп'ютерних напрямків, яка повністю базується на відкритому, безкоштовному програмному забезпеченні. При цьому якість та можливості перелічених програмних продуктів жодним чином не погіршують звичних можливостей наявного забезпечення (на базі ОС Windows).

Інформативна складова не суперечить існуючим навчальним програмам, бо суть залишається та сама, а міняється підхід, програмна платформа та інструментарій розв'язання задач, що підтверджується наведеними прикладами.

Також слід пам'ятати, що такий перехід виключає можливі майбутні закиди з приводу нелегітимності систем, що використовуються, в тому числі й в навчальному процесі.

ЛІТЕРАТУРА

1. Как Мюнхен перевёл 15 000 ПК с Windows на Linux (http://habrahabr.ru/post/222511/).
2. http://178.219.93.18:8080/Portal/WWW/prep_list.php?id_fac=7&id_dep=18.
3. http://www.ubuntu.com/.
4. http://www.linuxmint.com/.
5. http://178.219.93.18:8080/Portal/Data/7/18/7-18-lr8.pdf.
6. http://maxima.sourceforge.net/.
7. https://www.gnu.org/software/octave/.
8. http://www.scilab.org/.
9. http://ru.smath.info/.
10. http://www.lazarus.freepascal.org/.
11. https://eclipse.org/downloads/packages/eclipse-ide-cc-developers/lunasr1a.
12. http://www.freebasic.net/.
13. http://www.starbasic.net/.
14. http://178.219.93.18:8080/Portal/Data/7/18/7-18-lr9.pdf.

Надійшла до редколегії 03.02.2015.

УДК 004.054

ТИМОШЕНКО Д.В., к.т.н., ст. преподаватель
ШУМЕЙКО А.А., д.т.н., профессор,
ЖУЛЬКОВСКИЙ О.А., к.т.н., доцент

Днепродзержинский государственный технический университет

ОБУЧЕНИЕ НАЧАЛЬНЫМ НАВЫКАМ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Введение. Тестирование программного обеспечения (ПО) представляет собой исследование, проведенное с целью обеспечения заинтересованных сторон (заказчика и разработчика) объективной информацией о качестве продукта или услуги [1-5]. Другой целью тестирования ПО может быть оценка рисков реализации и внедрения тестируемого программного продукта.

Методы тестирования (испытаний) включают в себя процесс выполнения программы или приложения с целью найти ошибки (или другие дефекты) в ПО, а также оценить одно или несколько его свойств, например:

- отвечает ли требованиям заказчика;
- правильно ли реагирует на все виды входных данных;
- выполняет ли свои функции в пределах допустимого времени;
- достаточно ли удобно;
- достигает ли результата, требуемого заинтересованными сторонами.

Поскольку число возможных тестов для даже самых простых программных компонентов практически бесконечно, все методы тестирования ПО используют некоторую стратегию выбора тестов для имеющегося времени и ресурсов. Процесс тестирования является итерактивным, т.к. когда ошибка будет исправлена, она может осветить и другие, более глубокие ошибки, или может даже создать новые.

Тестирование ПО представляет собой важный элемент производственного цикла разработки программных продуктов, но до сих пор недостаточно представлен в процессе подготовки специалистов в области программной инженерии.

Постановка задачи. Учитывая, что профессия тестировщика стала одной из самых востребованных на рынке IT, очень важно при подготовке специалистов уделить достаточное внимание этому сегменту разработки программных продуктов. Достаточно большая часть выпускников-программистов свою первую работу выполняют в качестве тестировщика ПО, и поэтому работодатель заинтересован в специалистах с имеющимся багажом конкретных практических навыков.

Поиску решений этой проблемы посвящена данная работа.

Результаты роботы. Что ожидает работодатель от начинающего тестировщика? Самое главное, чтобы тестировщик не только нашел дефект, но и правильно его описал. Это базовый навык, т.к. обычно начинающий тестировщик работает с уже разработанным наставником тест-планом и ему нужно лишь, отталкиваясь от него, отыскивать ошибки ПО. Здесь неважно, как дефект был найден. Важно, чтобы он был однозначно описан и программист смог его воспроизвести с минимальными дополнительными вопросами.

Используемый подход основывается на движении от практики к теории. Его цель – дать студентам на себе прочувствовать, что же такое неправильно описанный дефект.

На первом этапе студенты должны протестировать без спецификации конкретное ПО и сделать отчет о найденных дефектах, при этом не требуется никаких уточнений о понятии «дефект», «отчет о дефектах» и т.п. Для проверки квалификации тестировщика предлагается использовать приложение ListBoxer (рис.1) [6], которое является в определенном смысле стандартом в этой сфере. Это приложение представляет собой программу, в которой реализован простейший, но достаточно всеохватывающий функционал с большим количеством ошибок, которые сделаны преднамеренно.

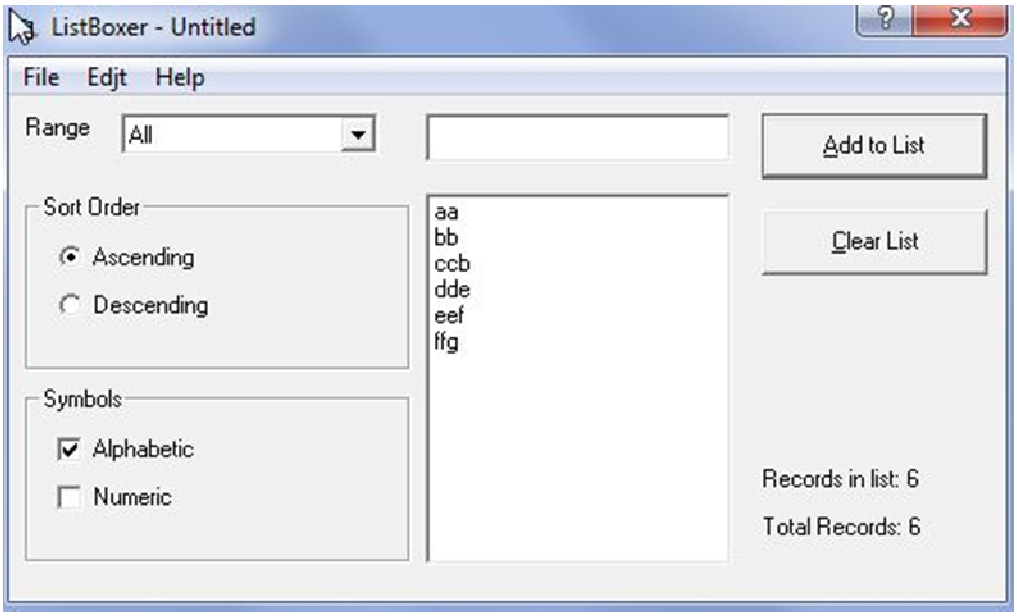

Рисунок 1 – Окно приложения ListBoxer

Суть приложения в том, что пользователь создает некий список из строк, которые могут состоять из символов латиницы и цифр, и проводит с этим списком различные манипуляции:

– добавляет новые элементы согласно правилам валидации;
– фильтрует по диапазону;
– упорядочивает по возрастанию/убыванию;
– показывает только латиницу или только цифры;
– сохраняет в файл или загружает из файла.

В виду того, что никаких требований к приложению ListBoxer нет, то предполагается, что тестирование опирается на здравый смыл тестировщика и на информацию из раздела «Помощь». Применяемая стратегия тестирования – «черный ящик», т.к. доступа к коду не имеется.

Несмотря на то, что приложение на вид простое, оно позволяет проработать такие подходы к функциональному тестированию, как:

– классы эквивалентности;
– классы граничных точек (предпочтительно вынести их в отдельный класс);
– попарное тестирование;
– таблица решений;
– сценарии пользователя.

У приложения есть инсталлятор, поэтому большим участком работ является тестирование инсталлятора. Отдельное внимание можно уделить тестированию удобства пользования (*usability*) и тестированию документации (помощи).

Ориентировочной цифрой приличного результата является нахождение тестировщиком 50 дефектов. Лучший студенческий результат показал 32 дефекта.

Почему же так важно правильно описать дефект? Дело в том, что стоимость неправильно описанной проблемы очень велика. Типичным сценарием в этом случае является «перекидывание» дефекта между программистом и тестировщиком. Квалифицированный программист будет отправлять назад дефект до тех пор, пока его содержимое не станет однозначно воспроизводить проблему.

Не стоит ожидать от студентов отличных результатов по тестированию, а тем более описанию дефектов в первый раз. Предлагается выносить на всеобщее обсуждение каждый отчет, чтобы все остальные могли представить себя на месте программиста, получившего ошибку.

Главные моменты, которые нужно донести о дефекте.

1. Дефект – это несоответствие фактического поведения ПО ожидаемому. Если фактический результат не равен ожидаемому – это дефект. В случае ListBoxer (на самом деле, это характерно и для многих коммерческих продуктов) ожидаемый результат берется из здравого смысла в широком понимании (сюда же добавляется опыт пользователя, стандарты в этой сфере и т.п.).
2. Дефект должен однозначно воспроизводиться. Должно быть приведено минимальное количество шагов, которое позволяет программисту повторить то, что видит тестировщик.
3. Версия тестируемого ПО. Дефект всегда обнаруживается в какой-то конкретной версии программы и исправляется в какой-то другой конкретной версии. Если у приложения нет версии – это дефект, который не позволяет тестировать, т.к. невозможно обеспечить учет и контроль ошибок.
4. Приоритет. Все дефекты могут быть как очень важными, так и совсем незначительными. Это зависит от того, какие требования выставлены к программному продукту. В отдельных случаях и опечатка может быть критическим дефектом (например, опечатка в основном номере телефона онлайн-магазина).

5. Автор дефекта. Это нужно, прежде всего, для взаимодействия в команде.

6. Идентификатор. Все дефекты должны иметь уникальные идентификаторы. Практически всегда для учета дефектов используются специальные системы, которые автоматически присваивают идентификатор.

Далее отметим типичные ошибки, которые совершают начинающие тестировщики при использовании ListBoxer.

1. Не указывается версия ListBoxer. Ни один студент не указал версию при оформлении дефектов в первый раз.

2. Не используется понятие «фактического результата» и «ожидаемого результата» (в привычной английской терминологии *actual*, *expected*). Вместо этого используют слова и выражения «неправильно», «неверно». Например, у каждого второго студента был дефект «неправильная сортировка или вводим данные, сортирует неправильно». Без подробностей, что такое неправильная сортировка, как будто у всех одинаковое представление. Нужно отметить, что этой проблеме (неожиданно для самих студентов) уделено чрезвычайно много времени, поскольку мнения разделились, что же такое «неправильная сортировка», если учитывать, что в строках присутствуют числа. Этот дефект – замечательный пример для того, чтобы продемонстрировать, как важно тестировщику довести однозначно программисту проблему, после чего можно перевести диалог в плоскость «это не дефект, это так и должно быть или же да, это дефект». Главное, чтобы программист правильно понял, на какую ситуацию ему указывают.

Другой случай из этого же ряда – при вводе запрещенных символов компьютер «пищит». Можно сказать, что фактический результат понятен. А вот какой же ожидаемый? Компьютер выдает сообщение с предупреждением о несоответствии формату, «пищит» как-то иначе и т.д.?

3. Не используются конкретные примеры вводимых данных и конкретного ожидаемого результата для них. Так, например, описание «неправильная сортировка» нужно разворачивать в подобный вид:

– запустить ListBoxer.exe(v.1.98);
– выбрать режим Numeric;
– ввести в текстовое поле: 27, 10, 300, 80;
– выбрать сортировку Ascending.

Фактический результат: 10, 27, 300, 80.

Ожидаемый результат: 10, 27, 80, 300.

4. Не используются скриншоты (например, рис.2). Конечно, необходимым условием описания дефекта является наличие всех шагов для воспроизведения. Скриншот не является обязательным по определению, но с точки зрения человеческого фактора наличие правильно сделанного скриншота существенно увеличивает скорость понимания проблемы. Особенно это касается дефектов, которые связаны с элементами интерфейса, такими как опечатки, «съехавшая» верстка в веб-проектах, элементами интерфейса, не соответствующим спроектированному дизайну и т.д.

5. Пропускается шаг с описанием, что именно нужно запустить, и сразу выполняется переход к месту, которое не работает. Во-первых, в этом случае не понятно, что запускается – инсталлятор или само приложение (это становится понятным дальше из контекста), во-вторых, вместе с этим шагом часто пропускается и указание версии, в которой найдена проблема.

6. Не пишется однотипно «фактический» и «ожидаемый» результат. Например, вместо такого описания:

Фактический результат: в выпадающем списке «Range» имеется пустое поле.

Ожидаемый результат: отсутствие пустого поля.

лучше воспринимается, когда ожидаемый результат будет описан в такой же последовательности, как и фактический:

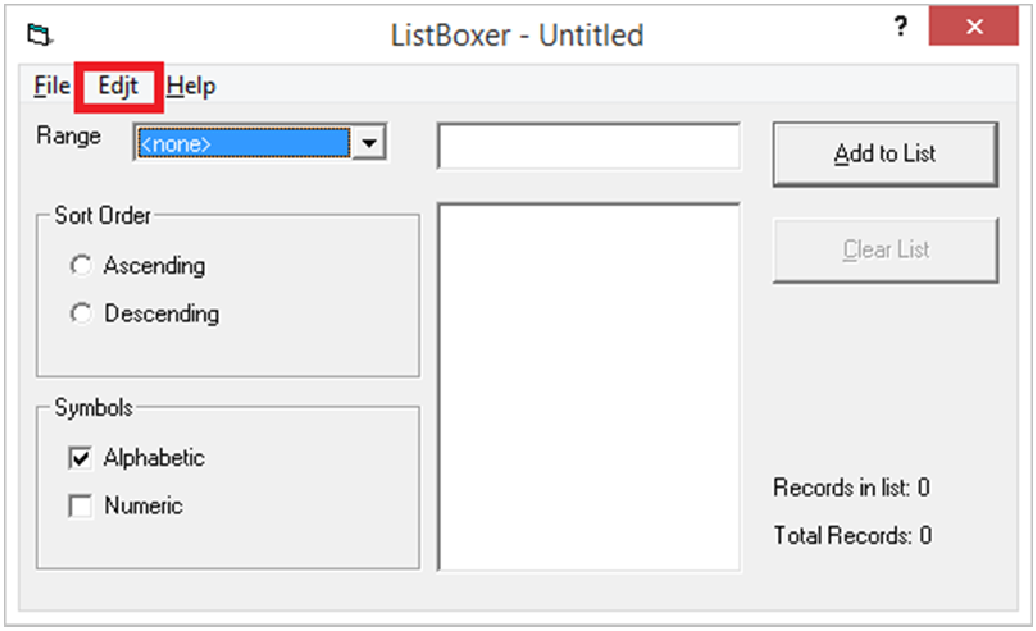

Рисунок 2. – Пример скриншота ListBoxer и его оформления

Фактический результат: В выпадающем списке «Range» имеется пустое поле.

Ожидаемый результат: В выпадающем списке «Range» нет пустого поля.

7. Не присваивается идентификатор дефекту. Важно обратить на это внимание, но при этом следить, чтобы при переносе дефектов из простого отчета в систему учета задач в названии проблемы не писали идентификаторы, т.к. они будут присваиваться системой автоматически.

8. В фактическом результате описывается и ожидаемый. Здесь типичное слово – «вместо», например:

Фактический результат: В пункте меню написано Edjt вместо Edit.

Ожидаемый результат: Edit.

Нужно так:

Фактический результат: В пункте меню написано Edjt.

Ожидаемый результат: В пункте меню написано Edit.

9. Используются лишние шаги. Характерным случаем является ошибка чтения из файла. В этом случае вместо того, чтобы приложить файл, чтение которого приводит к ошибке, тестировщик предлагает сначала воспроизвести сценарий получения данного файла, его сохранения и только потом открытия.

Пример с лишними шагами.

Запустить ListBoxer.exe (v.198);

1. Добавить в список значения 12, 45, doc, tatata.
2. С помощью пункта меню File-Save сохранить файл.
3. Открыть сохраненный файл.

Фактический результат: В списке отобразились: 45, doc, tatata (утерян первый элемент из файла).

Ожидаемый результат: В списке отобразились 12, 45, doc, tatata.

Нужно так:

1. Запустить ListBoxer.exe (v.198);
2. С помощью пункта меню File-> Open выбрать файл nofirst.lbx (приложен к дефекту).

Фактический результат: В списке отобразились: 45, doc, tatata (утерян первый элемент из файла).

Ожидаемый результат: В списке отобразились 12, 45, doc, tatata.

Помимо того, что первый вариант заставляет программиста делать лишние шаги, так он (что более важно) вводит в заблуждение, что на дефект влияет именно последовательность «создать данные->сохранить файл->открыть файл» в то время, как дефект лежит только в плоскости «открыть файл».

10. В одном дефекте описывается несколько похожих случаев. Таким примером служит ситуация с пунктами меню Copy, Cut. Действительно, ожидаемое поведение для стандартных функций Copy и Cut похоже, и если дефект выглядит однотипно, то кажется допустимым описать все одним дефектом. Но это в корне неверно, т.к. непосредственно в коде может быть два разных участка, в которые нужно внести изменения. В этой ситуации нужно заводить столько дефектов, сколько функций не работает согласно спецификации, а в самих дефектах будет полезным указать примечание, в котором указать, с какой подобной ситуацией столкнулся тестировщик.

Пример некорректного описания дефекта:

Фактический результат: Элементы меню Edit->Cut, Edit->Copy недоступны.

Ожидаемый результат: Edit->Cut и Edit->Copy доступны

Нужно писать так:

Фактический результат: Элемент меню Edit->Cut недоступен.

Ожидаемый результат: Элемент меню Edit->Cut доступен.

Примечание: Похожая проблема с элементом меню Edit->Copy (дефект № – ссылка на дефект).

Выводы. Таким образом, были представлены 10 наиболее распространенных ошибок в описании дефектов. Моделирование рабочей ситуации, когда начинающий тестировщик идет по разработанному тест-плану и заполняет сам дефекты, позволило каждому студенту проанализировать качество найденных дефектов и понять на собственном опыте, насколько усложняют рабочее взаимодействие ошибки на начальном уровне. Кроме этого, такое погружение в практику на первых занятиях стимулирует студентов более вдумчиво подходить к последующей теоретической части проектирования тестовых сценариев. В целом, использование рассмотренной методики обучения студентов-программистов при изучении дисциплины «Качество программного обеспечения и тестирование» позволило привить им начальные навыки успешного тестирования ПО, что даст возможность выпускникам повысить шансы на успешное трудоустройство и дальнейший карьерный рост.

ЛИТЕРАТУРА

1. Криспин Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд / Л.Криспин, Д.Грегори. – М.: Вильямс, 2010. – 464с.
2. Канер С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / С.Канер, Д.Фолк, Е.Кек Нгуен. – Киев: ДиаСофт, 2001. – 544с.
3. Калбертсон Р. Быстрое тестирование / Р.Калбертсон, К.Браун, Г.Кобб.— М.: Вильямс, 2002. – 374с.
4. Синицын С.В. Верификация программного обеспечения / С.В.Синицын, Н.Ю.Налютин. – М.: БИНОМ, 2008. – 368с.
5. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем / Б.Бейзер. – СПб.: Питер, 2004. – 320с.
6. IC Group, Inc. ListBoxer. [Электронный ресурс] – Режим доступа: https://www.listbox.com/.

Поступила в редколлегию 03.09.2015.