УДК [377.169.34:621.869]:004.415.53

Шкляева А. В., Полевщиков И. С.

ПРИМЕНЕНИЕ ИНТЕГРАЦИОННОГО И СИСТЕМНОГО ТЕСТИРОВАНИЯ ПРИ РАЗРАБОТКЕ ТРЕНАЖЕРНОГО КОМПЛЕКСА ДЛЯ ОБУЧЕНИЯ ОПЕРАТОРОВ ПОРТАЛЬНЫХ КРАНОВ

Анастасия Владиславовна Шкляева

магистр

ana.shusharina@gmail.com

Иван Сергеевич Полевщиков

кандидат технических наук, доцент

i.s.polevshchikov@mail.ru

ФГБОУ ВО Пермский национальный исследовательский политехнический университет, Россия, Пермь

INTEGRATION AND SYSTEM TESTING IN THE DEVELOPMENT OF TRAINING COMPLEX FOR PORTAL CRANES OPERATORS

Anastasia Vladislavovna Shkliaeva

Ivan Sergeevich Polevschikov

Perm National Research Polytechnic University, Russia, Perm

Аннотация. *Целью текущей работы является представление результатов тестирования разработанного тренажерного комплекса для обучения операторов портального крана. В процессе работы были использованы методы модульного, интеграционного и системного тестирования. В результате тестирования был проверен функционал тренажерного комплекса и выявлен ряд недочетов, требующих доработки. Работа выполнена при финансовой поддержке РФФИ, проект № 18-38-00835*

Abstract. *The aim of the current work is to present the test results of the developed simulation complex for the gantry crane operators training. During the work, unit, integration and system testing were used. As a result of testing, the functionality*

of the training complex was checked and a number of errors were identified that required fixing. The reported study was funded by RFBR according to the research project № 18-38-00835.

Ключевые слова: *тестирование, тренажерный комплекс, Unit-тесты, портальный кран, симулятор.*

Keywords: *testing, training complex, unit test, gantry crane, simulator.*

Тестирование является важной частью разработки программного обеспечения (ПО). Оно позволяет оценить корректность и ожидаемость результатов работы, как отдельных модулей создаваемого ПО, так и системы в целом.

В зависимости от степени изолированности компонентов, выделяют следующие виды тестирования: модульное (unit), компонентное, интеграционное и системное [1]. Также, в зависимости от степени автоматизированности, можно выделить ручное, автоматизированное и полуавтоматическое тестирование. В зависимости от масштаба программного обеспечения подбирается стратегия тестирования. В данной статье будут описаны результаты тестирования тренажерного комплекса (ТК) оператора портального крана, разработанного на основе симулятора AnyCrane [2].

Данный ТК представляет собой сложный программно-аппаратный комплекс, предназначенный для обучения операторов портальных кранов. В его состав входит 3 основные компонента: симулятор предметной области, имитирующий перегрузочный процесс и работу вспомогательного персонала; система обучения, включающая в себя алгоритмы оценки работы оператора и советующую систему; набор инструментов инструктора, необходимых для создания упражнений и отслеживания прогресса обучения.

Таким образом, было принято решение провести тестирование ТК в несколько этапов:

1) автоматизированное Unit-тестирование отдельных методов ПО;

2) полуавтоматическое интеграционное тестирование связок из нескольких модулей для совместной отработки комплексных функций;

3) ручное системное тестирование. Оно предполагает проверку корректности работы системы по разработанным тест-кейсам.

Для реализации процесса тестирования были прописаны тест-кейсы и их ожидаемые результаты. Для автоматизации процесса Unit и интеграционного тестирования был использован модуль TestRunner [3], встроенный в среду Unity, в которой производится разработка ТК.

Unit-тестирование

На момент проведения тестирования, ПО включало в себя 51 класс, обеспечивающий необходимый функционал системы. Для основной части методов классов были разработаны Unit-тесты. Рассмотрим несколько примеров от простого к более сложному.

Пример 1. Метод установки ограничения времени выполнения упражнения. Данная проверка является самой простой, т.к. в нем производится работа с приватной переменной через публичные методы.

На рисунке 1 представлен пример кода Unit-теста для тестового запуска Т2. В таблице 1 представлена сводная информация по всем проведенным для метода тестам.

```
[Test]
Ссылок: 0
public void Exercise_SetTimeLimit_10sec()
{
    //Создаем тестовую переменную
    Exercise testExercise = new Exercise("Test Exercise");
    //Задаем контрольное значение
    testExercise.restrictions.SetTimeLimit(10f);
    //Сравниваем контрольное значение с эталонным
    Assert.AreEqual(10f, testExercise.restrictions.GetTimeLimit());
}
```

Рисунок 1 — Пример программного кода для Unit-теста (Т2)

Таблица 1 — Результаты тестирования примера 1

№ запуска	Вход	Ожидаемый результат	Подтвердился ли результат
Т1	0f	Ограничение по времени не установлено	Да
Т2	10f	Ограничение по времени установлено на 10 секунд	Да
Т3	-1f	Вывод ошибки	Да

Пример 2. Метод расчета координат контрольных точек взаимодействия стропальщика с грузом. Это более сложное тестирование, предусматривающее проверку корректности работы алгоритма.

Для вызова алгоритма необходимо предварительно создать экземпляр класса Container. Класс Container имеет следующие атрибуты, на основании которых производится расчет координат контрольных точек - length, height, width.

На рисунке 2 представлен пример кода Unit-теста для тестового запуска T1. В таблице 2 представлена сводная информация по всем проведенным для метода тестам.

```
[Test]
Ссылок: 0
public void Container_CalculateInteractionPoints_40FeetContainer()
{
    //Создаем переменную для теста
    Container container = new Container(12.2f, 2.4f, 2.6f);
    var calculatedPointList = container.CalculateInteractionPoints();

    //Формируем правильный ответ
    Vector3 point1 = new Vector3(2.44f, 0, 0);
    Vector3 point2 = new Vector3(2.44f, 2.4f, 0);
    Vector3 point3 = new Vector3(6.1f, 2.4f, 1.3f);
    List<Vector3> rightPointList = new List<Vector3>() { point1, point2, point3 };

    //Сравниваем расчитанный ответ с правильным
    Assert.AreEqual(rightPointList, calculatedPointList);
}
```

Рисунок 2 — Пример программного кода для Unit-теста (T1)

Таблица 2 — Результаты тестирования примера 2

№ запуска	Вход	Ожидаемый выход	Подтвердился ли результат
T1	container {12,2; 2,4; 2,6}	[{2,4; 0; 0}, {2,4; 2,4; 0}, {6,1; 1,2; 1,3}]	Да
T2	container {6,1; 2,4; 2,6}	[{1,2; 0; 0}, {1,2; 2,4; 0}, {3,05; 1,2; 1,3}]	Да

Несколько методов классов, реализующих одну комплексную функцию, могут объединяться в модули. Всего в тренажерном комплексе содержится 16 модулей, реализующих его функционал, и модуль Интерфейса, предоставляющий возможность взаимодействия пользователя со всеми компонентами ТК.

Для тестирования модуля интерфейса была реализована методика сравнения эталонных и фактических изображений меню. Был подготовлен пул эталонных изображений для каждого окна меню. Тестирование производится следующим образом:

1) имитируется нажатие мыши на элемент меню (кнопку);

2) модуль Интерфейса обрабатывает событие перехода на другое окно меню;

3) происходит сравнение фактического и эталонного изображения окна меню.

Тест считается успешным в случае, если процент совпадения изображений равен 100 %.

Пример 3. Переход из окна авторизации в главное меню пользователя

На рисунке 3 представлен пример кода данного Unit-теста.

```
[UnityTest]
Ссылок: 0
public IEnumerator ClickOnLoginButtonAndPassingToMainMenu()
{
    //Загружаем начальную сцену и выполняем необходимые настройки
    SceneManager.LoadScene(0);
    yield return null;
    var eventSystemObject = GameObject.Find("UIManager(Clone)");
    var inputWrapper = eventSystemObject.AddComponent<FakeInputWrapper>();
    yield return null;
    //Устанавливаем позицию мыши на кнопку
    inputWrapper.SetMousePosition(508f, 301f);
    yield return null;
    //Кликаем левой кнопкой мыши
    inputWrapper.ClickLeftMouseButton();
    yield return new WaitForSeconds(1f);
    yield return new WaitForEndOfFrame();
    //Делаем скриншот экрана и сравниваем его с эталонным
    var scr = ScreenshotMaker.MakeScreenshot();
    string screenshotName = "UserMainMenu";
    var diff = ScreenshotMaker.CompareScreenshots(screenshotName, scr);
    yield return null;
    //Тест пройдет, если разница между скриншотами равна 0
    Assert.AreEqual(0f, diff);
}
```

Рисунок 3 — Пример программного кода Unit-теста

На рисунке 4 представлено эталонное (слева) и фактическое (справа) изображения.

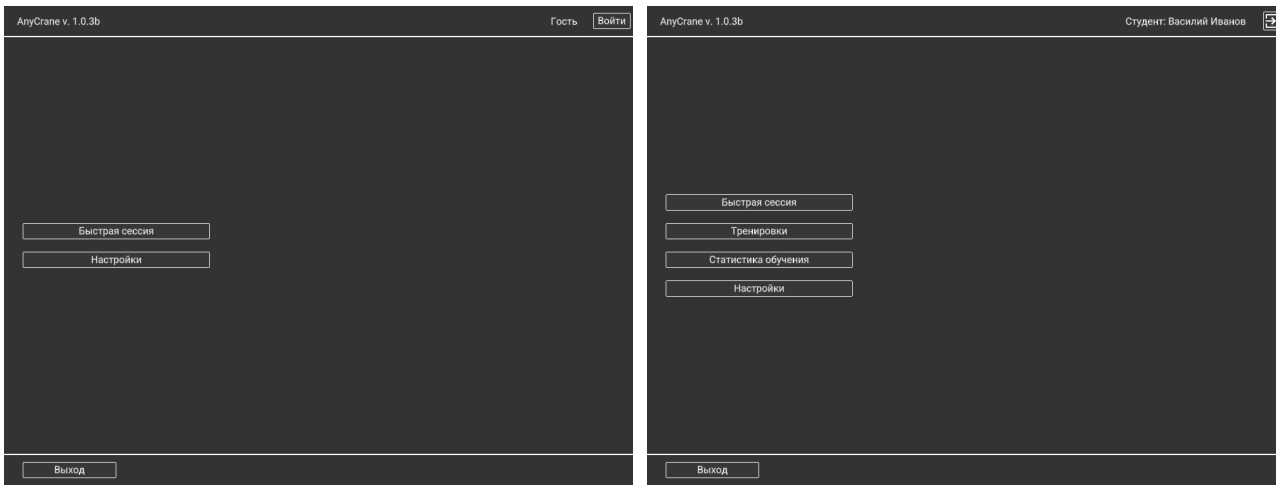

Рисунок 4 — Эталонное и фактическое изображения окна главного меню пользователя

В случае неуспешного прохождения тестирования, выводится окно как на рисунке 5.

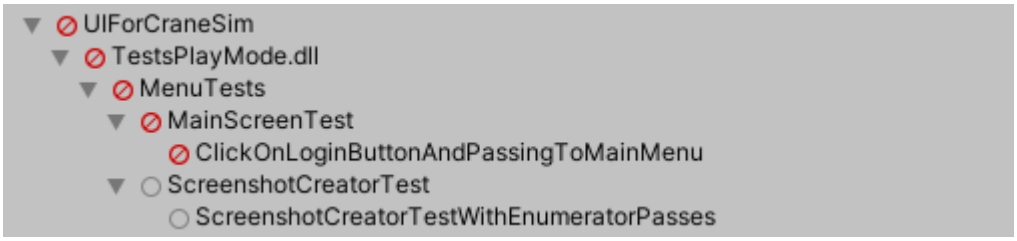

Рисунок 5 — Пример неуспешного прохождения теста

Таким образом было проверено 106 методов, а также 17 модулей, в суммарном объеме было проведено 360 тестовых запусков. Покрытие кода составило 75 %. Процент найденных ошибок составил 3.5 %.

Интеграционное тестирование

На данном этапе была проверена корректность совместной работы функционально связанных модулей. На рисунке 6 показана система взаимодействия модулей в ТК оператора портального крана.

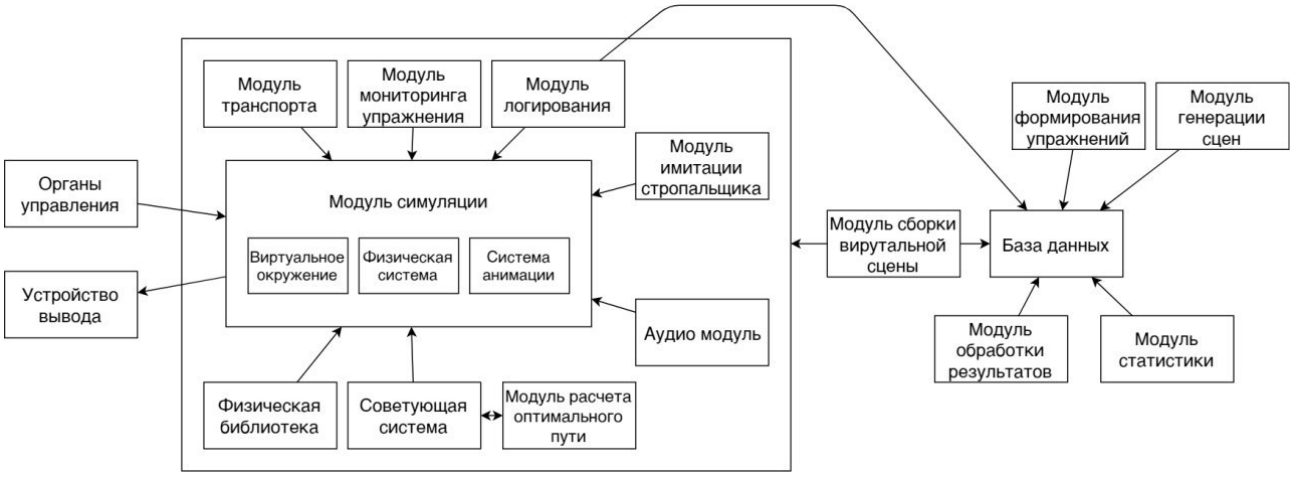

Рисунок 6 — Модульная архитектура ТК оператора портального крана

Рассмотрим несколько связок модулей в примерах 4 и 5.

Пример 4. Связка из модулей транспорта и симуляции. Проверим корректность работы метода расстановки транспорта на сцене.

На рисунке 7 представлен код Unit-теста описанной связки. В таблице 3 представлена сводная информация по всем проведенным тестам.

```
[UnityTest]
Ссылок: 0
public IEnumerator Transport_SetTruckOnScene_DefaultPosition()
{
    SceneBuildingManager.StartScene();
    yield return null;
    //Определяем позицию
    Vector3 truckPosition = new Vector3(25.2f, 0f, -5.2f);
    //Устанавливаем грузовой автомобиль в заданную позицию
    var truck = TransportManager.SetTruckOnScene("Man", truckPosition);

    //Тест считается выполненным, если заданная и фактическая позиции совпадают
    Assert.AreEqual(truckPosition, truck.transform.position);
}
```

Рисунок 7 — Пример программного кода Unit-теста связки модулей

Таблица 3 — Результаты тестирования примера 4

№ запуска	Вход модуля Транспорта	Ожидаемый выход из модуля Симуляции	Подтвердился ли результат
T1	{“Man”, {25,2; 0; -5,2}}	Грузовик с идентификатором Man будет расположен на сцене в позиции {25,2; 0; -5,2}	Да
T2	{“abcd”, {25,2; 0; -5,2}}	Вывод ошибки	Нет
T3	{“”, {25,2; 0; -5,2}}	Вывод ошибки	Да

Пример 5. Связка из модулей симуляции, логирования и базы данных (БД). Проверим корректность фиксации положения объекта на сцене в БД.

На рисунке 8 представлен код Unit-теста описанной связки.

```
[UnityTest]
Ссылок: 0
public IEnumerator SimulatingCargo_Logging_DB_NormalConditions()
{
    //Устанавливаем начальные настройки сцены
    var crane = SceneBuildingManager.AddActiveCrane("Ganz", Vector3.zero);
    var cargo = SceneBuildingManager.AddActiveCargo("Container_40ft");
    Vector3 initialAngle = new Vector3(0, 3.5f, 0);
    SceneBuildingManager.ConnectCraneAndCargo(crane, cargo, 15f, initialAngle);
    //Запускаем сцену
    SceneBuildingManager.StartScene();
    yield return null;
    //Запускаем модуль логгирования
    LogManager.StartWritingLog();
    yield return new WaitForSeconds(90f);
    //Запрашиваем статистику через 90 секунд после запуска имитации
    var result = DBManager.GetUserStatistics();

    //Т.к. запись в БД ведется 10 раз в секунду, тест будет считается
    //выполненным, если количество записей больше или равно чем 90 * 10
    Assert.IsTrue(result.Count >= 90 * 10);

}
```

Рисунок 8 — Пример программного кода Unit-теста связки модулей

На рисунке 9 представлены статистические данные по углу наклона груза, записанные за 90 секунд имитации и зафиксированные в базе данных.

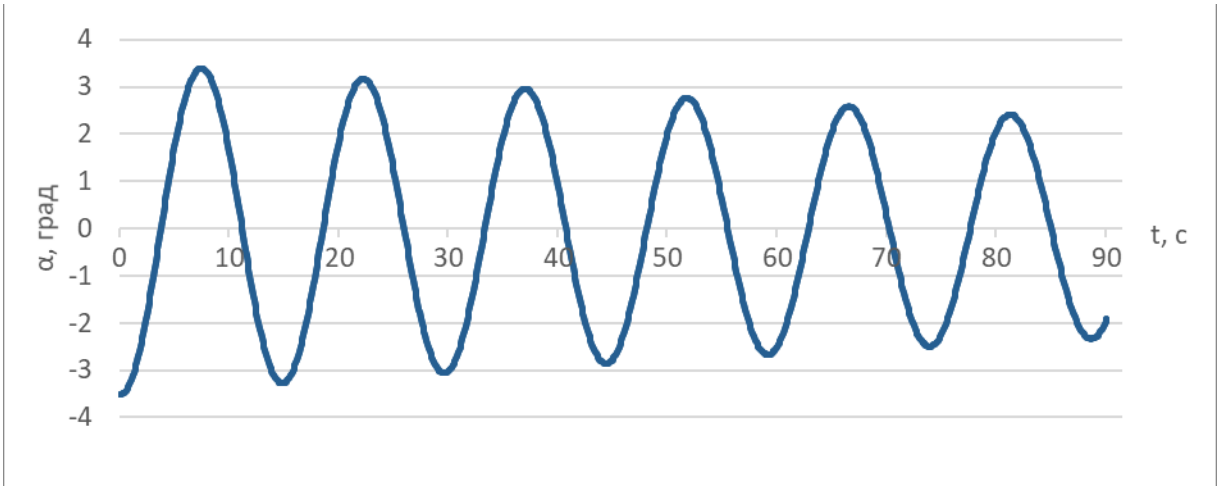

Рисунок 9 — Угол наклона груза

Всего на данном этапе было проверено 18 связок модулей, в суммарном объеме было проведено 30 тестов.

Системное тестирование

В рамках системного тестирования была произведена проверка интерфейсов, в зависимости от роли зарегистрировавшегося в системе пользователя. Для

этого мы зашли под каждым пользователем и провели соответствующие для роли тест-кейсы. Тестирование производилось в ручном режиме. Всего было проверено 14 тест-кейсов. Приведем несколько примеров.

Тест-кейс для обучаемого «Выполнение упражнения»:

1. В меню авторизации вводим учетные данные пользователя. Нажимаем кнопку «Войти».
2. Открывается окно меню пользователя. Выбираем пункт «Тренировки».
3. Открывается окно с перечнем доступных для выполнения тренировок. Выбираем тренировку «Перемещение груза». Нажимаем кнопку «Запустить».
4. Открывается виртуальная сцена. Выполняем упражнение в соответствии с показанным на экране заданием. После успешного завершения упражнения выводится окно с результатами выполнения упражнения. Нажимаем кнопку «Выход в главное меню».
5. Попадаем в главное меню пользователя, как на шаге 2.

В процессе тестирования ошибок не было выявлено.

Тест-кейс для инструктора «Создание упражнения»:

1. В меню авторизации вводим учетные данные инструктора. Нажимаем кнопку «Войти».
2. Открывается окно меню инструктора. Выбираем пункт «Сцены и упражнения».
3. Открывается окно с перечнем доступных действий. Выбираем «Создать новое упражнение».
4. Открывается окно с перечнем доступных сцен. Выбираем одну из сцен и зажимаем кнопку «Далее».
5. Открывается окно редактора упражнений. Добавляем 3 контрольные точки и расставляем их на сцене произвольным образом. Нажимаем кнопку «Сохранить упражнение».
6. Вводим название упражнения. Нажимаем кнопку «Ок». Попадаем в меню «Сцены и упражнения», как на шаге 3.

В процессе тестирования была сформирована следующая рекомендация: из окна «Сцены и упражнения» убрать кнопки «Создать упражнение» и «Создать сцену»; вместо «Редактировать упражнение» и «Редактировать сцену» сделать «Упражнения» и «Сцены» соответственно; функционал создания сцен и упражнений внести внутрь окон «Упражнения» и «Сцены» соответственно в виде кнопки «+»; после создания нового объекта возвращаться обратно в список доступных объектов, а не в окно «Сцены и упражнения», как это реализовано на данный момент.

Заключение

В процессе работы было проведено тестирование тренажера оператора портального крана. На этапе Unit тестирования было протестировано 106 методов и 17 модулей, среди всех тестовых запусков, в 3,5 % выявлены ошибки. На этапе интеграционного тестирования было протестировано 18 связок модулей. Всего было проведено 30 тестов. На этапе системного тестирования было проведено исследование пользовательского интерфейса в зависимости от роли пользователя по 14 тест-кейсам. На основании тестирования был составлен ряд рекомендаций по улучшению интерфейса.

Таким образом был выявлен ряд недочетов, требующий доработки. Однако они не влияют на выполнение ТК его функциональных требований.

Список литературы

1. Пивень, А. А. Тестирование программного обеспечения / А. А. Пивень, Ю. И. Скорин // Системи обробки інформації. – 2012. – № 4 (1). – С. 56–58.

2. AnyCrane: Towards a better Port Crane Simulator for Training Operators / R. A. Fayzrakhmanov, I. S. Polevshchikov, A. F. Khabibulin, F. I. Shkliaev, R. R. Fayzrakhmanov // 15th International industrial simulation conference. – Warsaw? 2017. – С. 85–87.

3. Unity Test Runner – URL: https://docs.unity3d.com/2017.4/Documentation/Manual/testing-editortestsrunner.html/.