УДК 004.052.2

ТЕСТИРОВАНИЕ ИМИТАЦИОННЫХ МОДЕЛЕЙ

Гацко А.А., Медведев О.С., Коркин Л.Р.

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь

Научный руководитель: Меженная М.М. – канд. техн. наук, доцент, доцент кафедры ИПиЭ

Аннотация. Рассмотрено тестирование имитационных моделей, включая основные концепции и направления тестирования. Также представлена актуальность использования имитационных моделей при тестировании сложных систем.

Ключевые слова: тестирование, валидация, верификация, имитационных

Введение. Впервые термин «модульное тестирование» появляется в материалах *Wintersim* в 1988 году, однако термин применяется к тестированию среды моделирования, а не собственно модели. Конференция по тестированию программного обеспечения существует с 1988 года. Так, уже в 1988 году на этой конференции обсуждалась проблема организации параллельной разработки имитационной модели и «функциональных тестов» к ней.

Основная часть. Валидация модели – это процесс подтверждения того, что построенная модель достаточно хорошо представляет объект исследования. Иными словами, валидация модели отвечает на вопрос – правильная ли модель используется в исследовании.

Верификация модели – это процесс подтверждения того, что модель реализована в среде моделирования именно так, как спроектирована, без искажений. Верификация отвечает на вопрос – правильно ли разработана модель.

Тестирование модели – это процесс нахождения неточностей и ошибок модели. При тестировании на вход модели подаются заранее подготовленные наборы входных данных, а на выходе проверяется соответствие выходных данных ожидаемым результатам.

Модульное тестирование – термин, широко использующийся в разработке программного обеспечения, относится к проверке корректности отдельных программных модулей, обычно небольших – классов и функций. С одной стороны, во многих контекстах функциональное тестирование эквивалентно модульному тестированию. Однако функциональное тестирование может относиться к тестированию некоторого аспекта функциональности программного обеспечения (ПО), проявляющегося во взаимодействии нескольких модулей (классов или функций). В имитационном моделировании целесообразно использовать именно термин «функциональное тестирование», так как чаще всего тестируется именно корректность взаимодействия нескольких логических модулей.

Однако с возрастанием сложности модели, а также длительности ее активного использования и доработки, сложность поддержания корректности модели возрастает. Так, например, при добавлении в модель нового аспекта поведения агента часто требуется проверить корректность взаимодействия добавляемого аспекта с уже реализованными. По результатам такой проверки часто выясняется, что логика уже реализованных аспектов поведения также нуждается в корректировке. Корректировка уже реализованной логики может привести к внесению в нее ошибок. Такие ошибки выглядят особенно нелепо с точки зрения конечного пользователя модели – после добавления новых аспектов поведения старые сценарии, в которых вроде бы нет таких новых аспектов, начинают почему-то исполняться с ошибками.

Усугубляет ситуацию то, что ошибки могут быть внесены не в логику поведения отдельных агентов, а в порядок взаимодействия агентов в определенной ситуации. В таком случае ошибка может обнаружиться много позже того момента, когда приводящая к ней логика была реализована, и затраты времени на ее исправление значительно возрастают.

Хорошим практическим решением описанной проблемы является создание набора тестов, подтверждающих, что работа уже реализованной логики модели соответствует ожиданиям. В тестировании ПО этот подход называется регрессионным тестированием. На практике становится ясно, что тесты логично добавлять непосредственно после реализации соответствующей функциональности. В этом случае разработчик модели наиболее полно погружен в контекст разрабатываемой функциональности, и для создания тестов требуется минимум затрат времени. Кроме того, успешное выполнение теста служит своеобразным критерием приемки разработанной или доработанной функциональности.

О том, как реализовать тестирование в коммерческой разработке, написано немало книг, и в области ИМ эта задача тоже не нова. Так, например, в публикациях конференции *WinterSim* 1995 года затрагиваются аспекты автоматизированного тестирования ИМ и предлагаются некоторые решения. К решению задачи тестирования ИМ в среде *AnyLogic* уже прибегали, но удовлетворительного решения для автоматизации тестирования предложено не было. Данная статья предлагает конфигурацию ИМ для организации ее тестирования в *AnyLogic*.

Структура проекта в *AnyLogic*

Предлагаемая структура проекта содержит четыре модуля, каждый из которых реализован в виде отдельного файла модели *AnyLogic*:

1. Модель данных (*DataModel*) – набор классов, описывающих предметную область. Корневой класс модели данных можно назвать «*Scenario*» – сценарий, содержащий всю необходимую информацию о предметной области. Также этот модуль может содержать классы, отвечающие за считывание / сохранение модели данных из файла / базы данных, а также за проверку корректности этих данных.

2. Модель (*ModelUnderTest*) – имитационная модель, которую необходимо протестировать. Эта модель не должна содержать простой эксперимент (*Simulation)* для запуска. По возможности она также не должна содержать анимацию, связанную со сбором данных для визуализации и статистики. Иными словами, важно как можно более строго соблюдать принцип отделения логики модели от ее внешнего представления. Эта имитационная модель будет запускаться с помощью JUnit-тестов без анимации и не будет содержать логику анимации.

3. Модель с анимацией (*AnimationModel*) – имитационная модель, содержащая анимацию и статистику. Этой моделью можно пользоваться для запуска конечного приложения и для запуска анимации при анализе корректности работы модели при тестировании.780 Секционные доклады ИММОД - 2021 458

4. Среда тестирования (*UnitTests*) – проект, содержащий все необходимое для организации процесса тестирования. Такая среда тестирования может использоваться как шаблон и для других проектов. Например, если создается тест для транспортной модели, к названию среды тестирования можно добавить префикс «*TransportModel*» (*TransportModelUnitTests*). Этот проект содержит классы с *JUnit*-тестами, а также ссылки на *Jar*-файлы (*a.jar* и *b.jar*), необходимые для запуска тестов.

Наличие тестов позволяет более уверенно добавлять новую функциональность, снижая риск внесения ошибок. Тестирование становится одним из ключевых способов управления сложностью, в особенности при командной разработке и создании моделей в несколько этапов.

Потребность в тестировании сложных моделей также косвенно подтверждается наличием в профессиональном сообществе следующих практик, заменяющих тестирование или служащих схожим с тестированием целям:

– создание «эталонных» сценариев, дающих ожидаемые результаты;

– анализ поведения модели при граничных значениях входных параметров;

– подсчет материального баланса и других контрольных значений.

Таким образом, потребность в тестировании заметно возрастает с ростом сложности и срока использования имитационной модели.

Потребность в тестировании имитационных моделей возрастает с ростом их сложности и срока использования, а встроенных средств функционального тестирования моделей в популярных инструментах ИМ нет. При создании моделей не в средах ИМ, а в инструментах программирования общего назначения имеется возможность использовать подходы к тестированию, являющиеся стандартными в индустрии разработки ПО. Однако тестирование имитационных моделей имеет свои особенности, отличающие его от тестирования ПО в целом. Эти особенности хорошо теоретически освещены в научной литературе, однако требуют осмысления и обобщения для создания практических рекомендаций по тестированию моделей.

Заключение. Практика показывает, что даже при наличии технических средств тестирования наиболее сложным аспектом тестирования ИМ является не разработка самих тестов, а проектирование моделей таким образом, чтобы ключевые алгоритмы модели можно было выделить в отдельные, независимые от других алгоритмов, модули.

Тем не менее, применение функционального тестирования позволяет разработчикам контролировать сложность даже больших ИМ и избегать снижения скорости разработки по мере роста сложности моделей. Наличие тестирования открывает дополнительные возможности по анализу логики моделей, такие как расчет степени покрытия тестами функциональности модели и оценка качества тестов с помощью мутационного тестирования.

Представляется целесообразным внедрение тестирования в практику работы разработчиков ИМ, в особенности в проекты по разработке больших и сложных имитационных моделей. Это, в свою очередь, требует наличия встроенных или добавления поддержки внешних средств тестирования в популярные инструменты и среды моделирования.

Список литературы

1. Особенности тестирования мобильных приложений [Электронный ресурс]. – 2019. – Режим доступа: https://qaevolution.ru/osobennosti-testirovaniya-mobilnyx-prilozhenij/– Дата доступа: 23.02.2022.

2. Тестирование имитационных моделей [Электронный ресурс]. – 2021. – Режим доступа: https://habr.com/ru – Дата доступа: 12.02.2022.

3. Научные статьи [Электронный ресурс]. – 2022. – Режим доступа: https://cyberleninka.ru/article – Дата доступа: 23.02.2022.

UDC 004.052.2

SIMULATION TESTING

Hatsko A.A., Medvedev O.S., Korkin l.R.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Mezhennaya M.M. – PhD, assistant professor, associate professor of the department of EPE

Annotation. The testing of simulation models is considered, including the basic concepts and directions of testing. The relevance of using simulation models when testing complex systems is also presented.

Keywords: testing, validation, verification, simulation