|  |  |
| --- | --- |
| ДИСЦИПЛИНА | **Методы верификации и валидации характеристик программного обеспечения** |
|  | (полное наименование дисциплины без сокращений) |
| ИНСТИТУТ | **информационных технологий** |
| КАФЕДРА | **математического обеспечения и стандартизации информационных технологий** |
|  | (полное наименование кафедры) |
| ВИД УЧЕБНОГО | **Материалы для практических/семинарских занятий** |
| МАТЕРИАЛА | (в соответствии с пп.1-11) |
| ПРЕПОДАВАТЕЛЬ | **Петренко Александр Анатольевич** |
|  | (фамилия, имя, отчество) |
| СЕМЕСТР | **3, 2023-2024** |
|  | (указать семестр обучения, учебный год) |

**Тестирование с помощью модели MBT (Model Based Testing)**

На основе изучения материала лекций по дисциплине «Методы верификации и валидации характеристик программного обеспечения» требуется выполнить следующее.

1. Выбрать поведенческую систему (это может быть приложение с логикой в виде состояний, взаимодействующие системы, диалоговые системы с несколькими режимами работы).
2. Создать модель в виде Cord-скрипта и упрощенной реализации на С#.
3. Сгенерировать тесты по модели.
4. Добавить функционал и проверить корректное выполнение тестов

**1. Выбор поведенческой системы**

В качестве примера поведенческой системы возьмем автомат продажи билетов на поезд. Система может иметь следующие состояния:

* Начальное состояние: ожидание пользователя.
* Состояние выбора пункта назначения.
* Состояние выбора типа билета (стандартный/льготный).
* Состояние оплаты.
* Состояние завершения (билет выдан).

**2. Создание модели в виде Cord-скрипта и упрощенной реализации на C#**

**Модель в виде Cord-скрипта**

Cord — это DSL (domain-specific language) для описания тестов, моделей и переходов между состояниями. Пример Cord-скрипта для системы автоматов продажи билетов может выглядеть следующим образом:

State Idle {

transition to SelectDestination on UserInput()

}

State SelectDestination {

transition to SelectTicketType on DestinationChosen()

}

State SelectTicketType {

transition to Payment on TicketTypeSelected()

}

State Payment {

transition to Complete on PaymentSuccess()

transition to Idle on PaymentFailure()

}

State Complete {

transition to Idle on TicketIssued()

}

В этом скрипте описаны состояния и переходы между ними на основе событий.

#### Упрощенная реализация на C#

Создадим простую реализацию на C#, которая моделирует этот процесс.

public enum TicketMachineState

{

Idle,

SelectDestination,

SelectTicketType,

Payment,

Complete

}

public class TicketMachine

{

public TicketMachineState State { get; private set; }

public TicketMachine()

{

State = TicketMachineState.Idle;

}

public void UserInput()

{

if (State == TicketMachineState.Idle)

{

State = TicketMachineState.SelectDestination;

}

}

public void DestinationChosen()

{

if (State == TicketMachineState.SelectDestination)

{

State = TicketMachineState.SelectTicketType;

}

}

public void TicketTypeSelected()

{

if (State == TicketMachineState.SelectTicketType)

{

State = TicketMachineState.Payment;

}

}

public void PaymentSuccess()

{

if (State == TicketMachineState.Payment)

{

State = TicketMachineState.Complete;

}

}

public void PaymentFailure()

{

if (State == TicketMachineState.Payment)

{

State = TicketMachineState.Idle;

}

}

public void TicketIssued()

{

if (State == TicketMachineState.Complete)

{

State = TicketMachineState.Idle;

}

}

}

### 3. Генерация тестов по модели

Тесты можно сгенерировать, проверяя корректность переходов между состояниями. Пример теста на C# для данной модели:

using Xunit;

public class TicketMachineTests

{

[Fact]

public void Test\_StateTransitions()

{

var machine = new TicketMachine();

// Начальное состояние должно быть Idle

Assert.Equal(TicketMachineState.Idle, machine.State);

// Переход к SelectDestination

machine.UserInput();

Assert.Equal(TicketMachineState.SelectDestination, machine.State);

// Переход к SelectTicketType

machine.DestinationChosen();

Assert.Equal(TicketMachineState.SelectTicketType, machine.State);

// Переход к Payment

machine.TicketTypeSelected();

Assert.Equal(TicketMachineState.Payment, machine.State);

// Успешная оплата — переход к Complete

machine.PaymentSuccess();

Assert.Equal(TicketMachineState.Complete, machine.State);

// Возврат в Idle после выдачи билета

machine.TicketIssued();

Assert.Equal(TicketMachineState.Idle, machine.State);

}

[Fact]

public void Test\_PaymentFailure()

{

var machine = new TicketMachine();

// Проходим до состояния оплаты

machine.UserInput();

machine.DestinationChosen();

machine.TicketTypeSelected();

// Провал оплаты — возврат в Idle

machine.PaymentFailure();

Assert.Equal(TicketMachineState.Idle, machine.State);

}

}

### 4. Добавление функционала и проверка выполнения тестов

Можно добавить новый функционал, например, проверку баланса пользователя перед оплатой, и обновить модель, чтобы включить проверку баланса:

public bool CheckBalance()

{

// Допустим, баланс всегда достаточен для этого примера

return true;

}

public void PaymentAttempt()

{

if (State == TicketMachineState.Payment)

{

if (CheckBalance())

{

PaymentSuccess();

}

else

{

PaymentFailure();

}

}}

Теперь тесты можно обновить для проверки нового функционала, чтобы убедиться, что они корректно выполняются.