Московский государственный технический университет
имени Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ
и информационные технологии»

Тестирование и отладка ПО

Лабораторные работы

Студент: Салем Б.Р.
Группа: ИУ7-71
Преподаватель: Рогозин О.В.

Москва, 2019

Содержание

1 Цель работы

Цель данной работы - протестировать приложение чат-бот для отображения ближайших концертов музыкальных исполнителей из списка подписок.

1.1 Описание тестируемой системы

Приложение чат-бот состоит из следующих частей:

а) сервер
б) база данных
в) клиентская часть

Сервер написан на языке C# с использованием API от ресурса KudaGo. Основное предназначение - обработка запросов от клиента.

База данных - реляционная СУБД MSSQL. Обращение к базе данных осуществляется с помощью Entity Framework.

Клиентская часть представляет собой приложение Telegram, связь с клиентом осуществляется с помощью API Telegram.Bot.

Основные сущности базы данных:

а) Users - таблица с пользователями
б) Groups - таблица с музыкальными исполнителями
в) Concerts - таблица с информацией о концертах
г) Subscriptions - таблица с подписками пользователей

program.cs - главный файл тестируемого приложения.

В файле **program.cs** поступают запросы от клиента, можно выделить следующие ключевые классы для обработки запросов:

а) UserActions.cs - обработка запросов, связанных с информацией о пользователе

б) GroupActions.cs - обработка запросов, связанных с информацией о музыкальных исполнителях

в) ConcertActions.cs - обработка запросов, связанных с информацией о концертах

г) CityActions.cs - обработка запросов, связанных с информацией о городе пользователя

д) SubscriptionActions.cs - обработка запросов, связанных с информацией о подписках пользователей

В клиентском приложении пользователь с помощью сообщения /start и после чего пользователь может использовать следующие команды:

а) /city 'город пользователя' - добавление города пользователя в таблицу пользователей для дальнейшего поиска концертов в этом городе

б) /add 'название исполнителя' - добавляет музыкального исполнителя в список подписок пользователя

в) /list - отображает подписки в виде пронумерованного списка

г) /show - отображает ближайшие концерты исполнителей в городе пользователя из его подписок

д) /remove 'название исполнителя' - удаляет исполнителя из подписок пользователя

е) /clear - очищает все подписки пользователя

1.2 Рассматриваемые виды тестирования

В данной работе будут рассмотрены следующие виды тестирования:

а) модульное

б) интеграционное

в) регрессионное

г) функциональное

д) автоматизированное

Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Регрессионное тестирование - это вид тестирования направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде.

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

2 Модульное тестирование

Цель модульного тестирования - изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. С помощью модульных тестов были протестированы основные компоненты приложения. Как было указано раннее, в базе данных существует 4 основные сущности - две из которых зависят от первичных ключей двух других таблиц. Поэтому было решено провести модульное тестирование двух независимых сущностей. Для каждой сущности существует класс для выполнение определенных действий с указанной сущностью.

2.1 Модуль GroupActions

Таблица 2.1 — Тестирование модуля GroupActions

Проверка наличия группы с название group1	
Входные данные	{groupid1: 104, groupid2:105, groupname1:group1, groupname2:group2}
Ожидаемый результат	
True Задействовано методов класса	2(InsertGroup, ContainsGroup)
Проверка отсутствия группы с название nogroup	
Входные данные	{groupid1: 104, groupid2:105, groupname1:group1, groupname2:group2}
Ожидаемый результат	
False Задействовано методов класса	2(InsertGroup, ContainsGroup)
Нахождение группы по id: 999	
Входные данные	{groupid: 999,groupname:group1,}
Ожидаемый результат	
True Задействовано методов класса	2(InsertGroup, FindGroupById)
Нахождение группы по groupname: group1	
Входные данные	{groupid: 999,groupname:group1,}
Ожидаемый результат	
True Задействовано методов класса	2(InsertGroup, FindGroupByName)
Mock тест: Удаления группы из таблицы Group	
Входные данные	3 объекта класса User с groupid и groupname. idtodelete для удаления
Ожидаемый результат	Количество объектов в таблице: 2 Сработано сохранений изменений в таблице: 1
Задействовано методов класса	1(DeleteGroup)

2.2 Модуль UserActions

Таблица 2.2 — Тестирование модуля UserActions

Mock тест: Вывод пользователей из таблицы User	
Входные данные	3 объекта класса User с userid и usercity
Ожидаемый результат	Вывод списка с 3-мя объектами класса User
Задействовано методов класса	1(SelectAllUsers)
Mock тест: Добавление пользователя в таблицу User	
Входные данные	3 объекта класса User с userid и usercity
Ожидаемый результат	Сработано добавлений в таблицу: 1 Сработано сохранений изменений в таблице: 1
Задействовано методов класса	1(InsertUser)
Mock тест: Нахождение пользователя по userid в таблице User	
Входные данные	3 объекта класса User с userid и usercity. findid для поиска
Ожидаемый результат	Результат метода не NULL
Задействовано методов класса	1(FindUser)
Mock тест: Изменение города пользователя в таблице User	
Входные данные	3 объекта класса User с userid и usercity. newcity для обновления города второго пользователя
Ожидаемый результат	значение newcity равен городу usercity второго пользователя
Задействовано методов класса	1(UpdateUser,SelectAllUsers для представления таблицы в виде списка))

2.3 Модуль KudaGoApi

Данный модуль отправляет POST-запрос для получения актуальной информации о концертах группы, которую мы указываем в параметрах POST-запроса. Не смотря на то, что данный API активно используется в приложении, протестировать его крайне сложно, т.к. его результаты зависят от происходящих концертах в реальной жизни, следовательно результаты тестов не постоянны. Но были протестированы 2 простых случая, в которых проверяется результат при попытки получить информацию о ближайших концертах с пустым и null именем.

Таблица 2.3 — Тестирование модуля KudaGoApi

Проверка наличия концерта группы ”	
Входные данные	{groupname = String.Empty}
Ожидаемый результат	null
Задействовано методов класса	1(GetAllConcertsByGroup)
Проверка наличия концерта группы ”	
Входные данные	{groupname = null}
Ожидаемый результат	null
Задействовано методов класса	1(GetAllConcertsByGroup)

2.4 Покрытие

Рисунок 2.1 — Покрытие после модульных тестов, включая Mock-тесты

3 Интеграционное тестирование

В данной части работы будет осуществлено тестирование модулей сервера, обрабатывающие запросы, взаимодействующие с базой данных, в данных тестах рассмотрены сценарии, в которых совместно используются несколько модулей.

3.1 Тестирование модулей

В тестах ниже используются методы основных классов упомянутых выше. Интеграционные тесты описаны в таблице 3.1.

Таблица 3.1 — Интеграционное тестирование

Удаление существующей подписки	
Запрос	{userid: 777, groupname: MyGroup, groupid: 111}
Ожидаемый результат	Удаление исполнителя из подписки
Используемые классы	UserActions,SubscriptionActions,GroupActions
Удаление несуществующей подписки	
Запрос	{userid:777, groupdid:111, wrongid:1111, groupname: NoGroup }
Ожидаемый результат	Список подписок остается тем же, функция удаления возвращает результат = null
Используемые классы	UserActions,SubscriptionActions,GroupActions
Вывод подписок пользователя	
Запрос	{userid: 999, groupid:222, groupname: RandGroup }
Ожидаемый результат	Вывод подписок пользователя с userid=999
Используемые классы	UserActions,SubscriptionActions
Удаление всех подписок одного пользователя	
Запрос	{userid: 777, groupname: MyGroup, groupid:111 }
Ожидаемый результат	даление подписок пользователя с id=777
Используемые классы	UserActions,SubscriptionActions,GroupActions
Удаление всех подписок одного пользователя, не затрагивая чужие	
Запрос	{userid: 777, additional userid: 787, groupname: MyGroup, groupid:111 }
Ожидаемый результат	Удаление подписок пользователя с id=777, подписки пользователя с id=787 остались прежними
Используемые классы	UserActions,SubscriptionActions,GroupActions
Добавление концерта новой группы	
Запрос	{userid: 777, groupname: RandGroup, groupid:888, Concert: EmptyConcert}
Ожидаемый результат	Добавление группы в таблицу групп и добавление концерта этой группы в таблицу концертов
Используемые классы	UserActions,SubscriptionActions,GroupActions, ConcertActions
Добавление концерта существующей группы	
Запрос	{userid: 777, groupid:888, Concert: EmptyConcert}
Ожидаемый результат	Добавление концерта группы в таблицу концертов
Используемые классы	UserActions,SubscriptionActions,GroupActions, ConcertActions

3.2 Покрытие

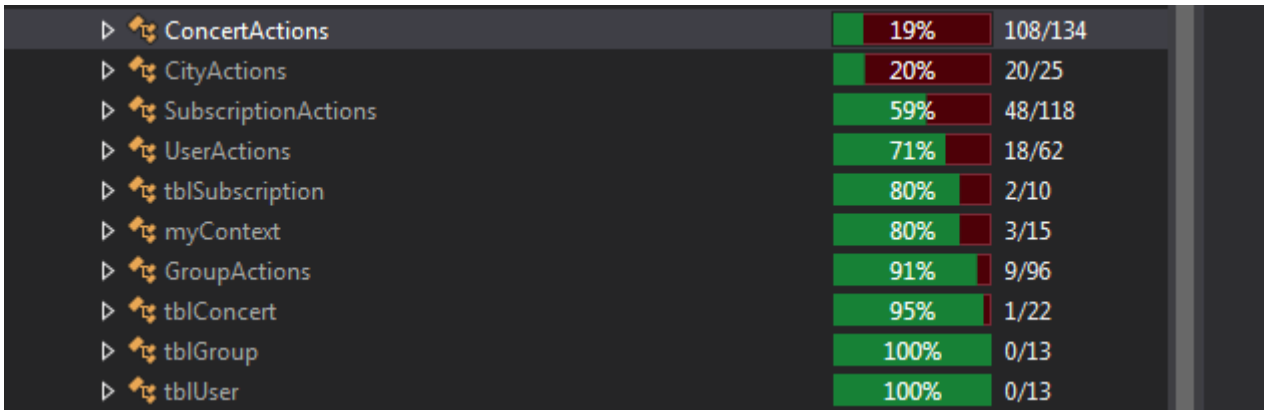

Рисунок 3.1 — Покрытие после модульных тестов и интеграционных

4 Регрессионное тестирование

Регрессионное тестирование — это выборочное тестирование, позволяющее убедиться, что изменения не вызвали нежелательных побочных эффектов, или что измененная система по-прежнему соответствует требованиям. Для того чтобы знать, какие тесты перезапускать после того или иного изменения в программе, нужно определить, от каких конкретно частей программы (модулей, методов, и т.п.) зависит результат каждого теста. Для этого часто используется управляющий граф , отображающий поток управления программы, по которому легко отследить зависимости одних блоков/модулей/методов от других.

Был построен один из вариантов управляющего графа: граф вызовов , показывающий, какие методы или функции вызывают какие. Граф был построен с помощью утилиты NDepend. Так как я хотел провести регрессионное вместе с автоматизированным, то было решено протестировать те части кода, которые не зависят от базы данных и могут быть выполнены на стороне удаленной виртуальной машины. Поэтому для регрессионного и автоматизированного тестирования были выбраны Mock-тесты.

Таблица 4.1 — Матрица вызовов Mock-тестов

	Методы			
Тесты	SelectAllUsers()	InsertUser()	FindUser()	DeleteGroup()
GetAllUsersTest	+	-	-	-
InsertUserTest	-	+	-	-
FindUserTest	-	-	+	-
TestDeletion	-	-	-	+
UpdateUserTest	+	-	-	-

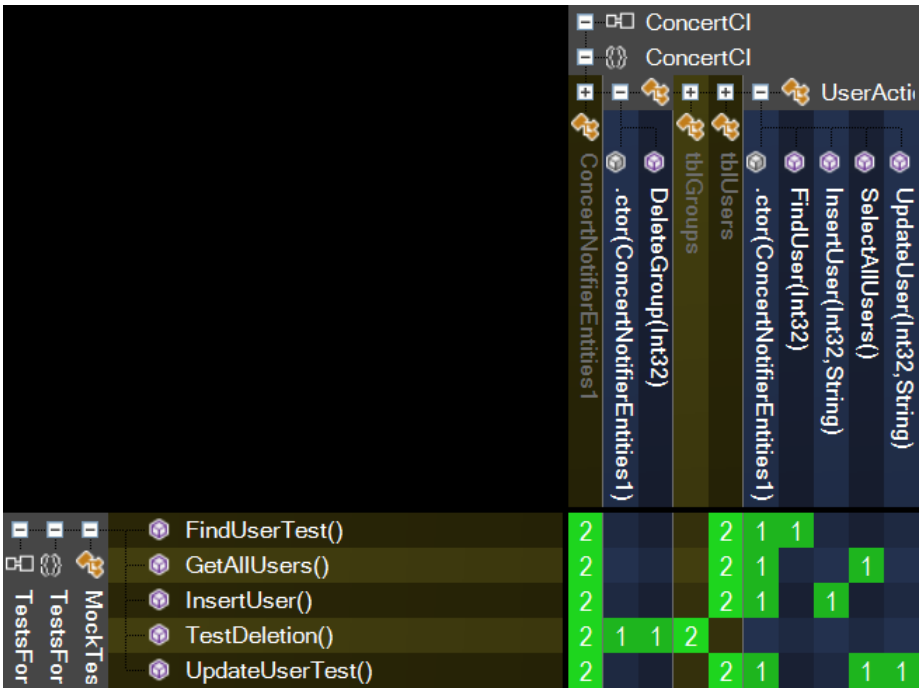

Рисунок 4.1 — Матрица вызовов для Mock-тестов, построенная NDepend

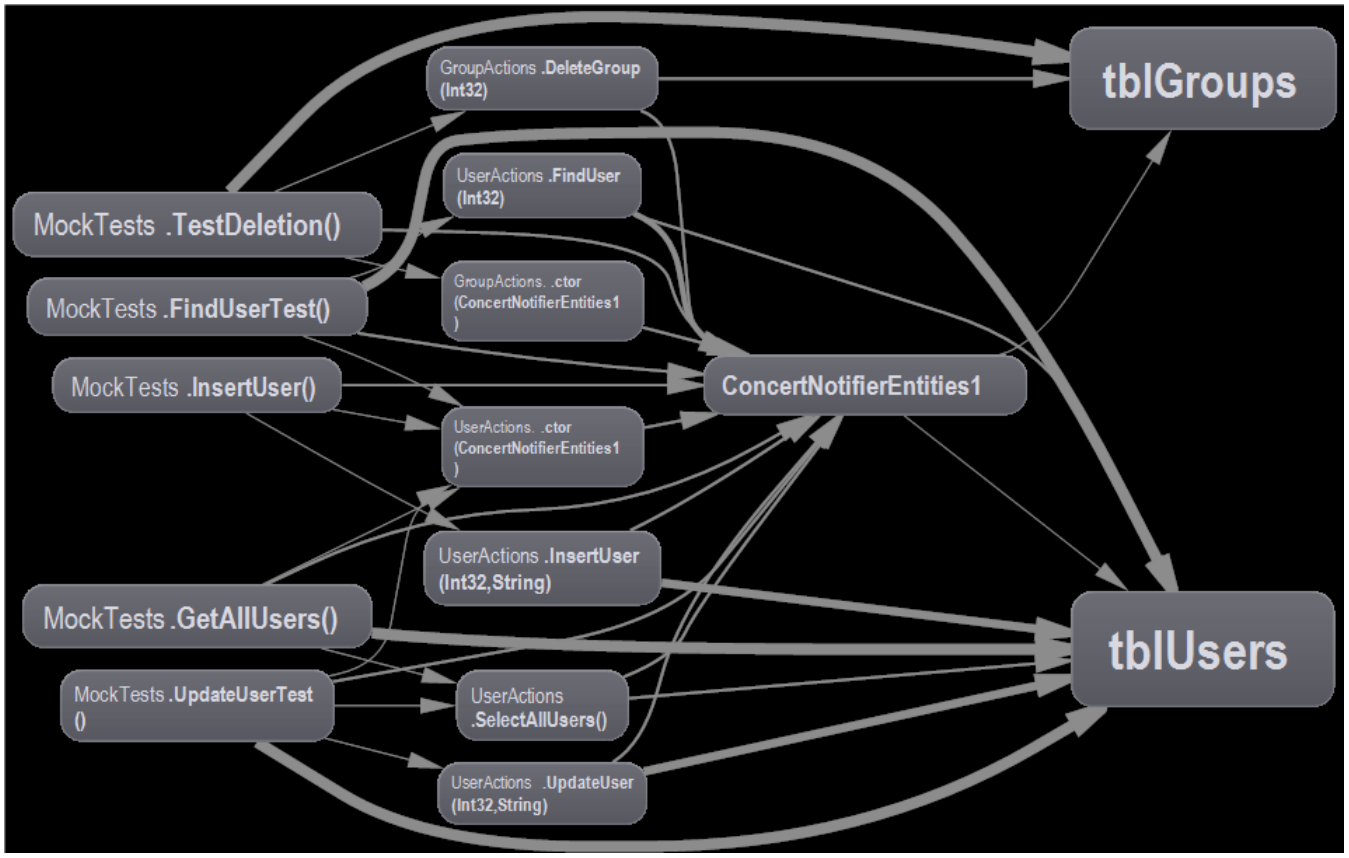

Рисунок 4.2 — Граф вызовов для Mock-тестов, построенный NDepend

В данном сценарии Mock-тестов мы изменили все методы, добавив обработку исключений с помощью оператора try-catch. Все тесты были успешно пройдены после добавления обработки исключений, поэтому было решено намерено допустить ошибку в методе DeleteGroup(), чтобы проверить случай, при котором не все тесты удачно пройдут автоматизированное тестирование. Подробнее это описано в следующем разделе под названием **"Автоматизированное тестирование"**

Было решено также построить матрицу вызовов для всех тестов, используя NDepend. Граф вызовов из-за высокого разрешения не был приложен к отчету, но может быть найден в папке с отчетом как и все остальные иллюстрации данного отчета.

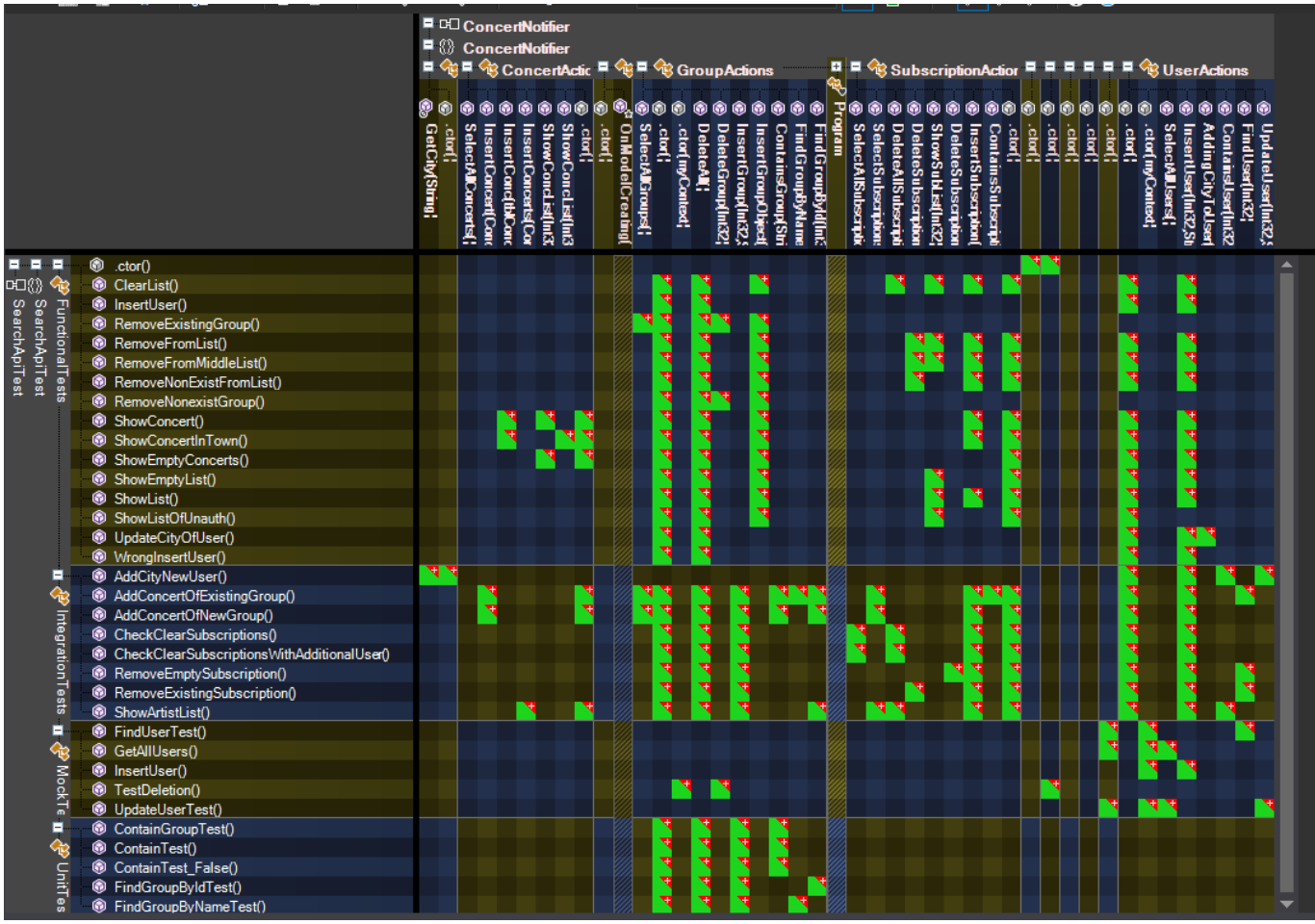

Рисунок 4.3 — Прохождение тестов до изменения блока K

5 Автоматизированное тестирование

Реализация тестов — достаточно затратный процесс, поэтому часто прибегают к тем или иным средствам, его облегчающим. В данной лабораторной работе для автоматизированного тестирования использовался ресурс **Travis CI**. Данный ресурс позволяет проверять работоспособность проекта, после каждого обновления репозитория на GitHub, запуская проект на удаленной виртуальной машине и выводя результаты в виде мини-отчета. Для этого проект с Mock-тестами был загружен на ресурс GitHub и к нему был подключен Travis CI и написан скрипт для сборки проекта.

Листинг 5.1 — Сценарий сборки для Travis CI

```
language: csharp
solution: ConcertCI.sln
before_install:
- sudo apt-get install nunit-console
before_script:
- nuget restore ConcertCI.sln
after_script:
- nunit-console TestsForCI/bin/Release/TestsForCI.dll
```

После этого был рассмотрен случай изменения метода DeleteGroup(), который был разобран в предыдущем разделе.

```
1711 $ nunit-console TestsForCI/bin/Release/TestsForCI.dll
1712 NUnit-Console version 2.6.3.0
1713 Copyright (C) 2002-2012 Charlie Poole.
1714 Copyright (C) 2002-2004 James W. Newkirk, Michael C. Two, Alexei A. Vorontsov.
1715 Copyright (C) 2000-2002 Philip Craig.
1716 All Rights Reserved.
1717
1718 Runtime Environment -
1719    OS Version: Unix 4.4.0.101
1720   CLR Version: 4.0.30319.42000 ( Mono 4.0 ( 5.18.0.225 (tarball Fri Dec 21 19:40:20 UT
1721
1722 ProcessModel: Default    DomainUsage: Single
1723 Execution Runtime: mono-4.0
1724 .The remote server returned an error: (400) Bad Request.
1725 .The remote server returned an error: (400) Bad Request.
1726 The remote server returned an error: (400) Bad Request.
1727 ....
1728 Tests run: 6, Errors: 0, Failures: 0, Inconclusive: 0, Time: 2.7400483 seconds
1729   Not run: 0, Invalid: 0, Ignored: 0, Skipped: 0
```

Рисунок 5.1 — Прохождение тестов до изменения метода DeleteGroup()

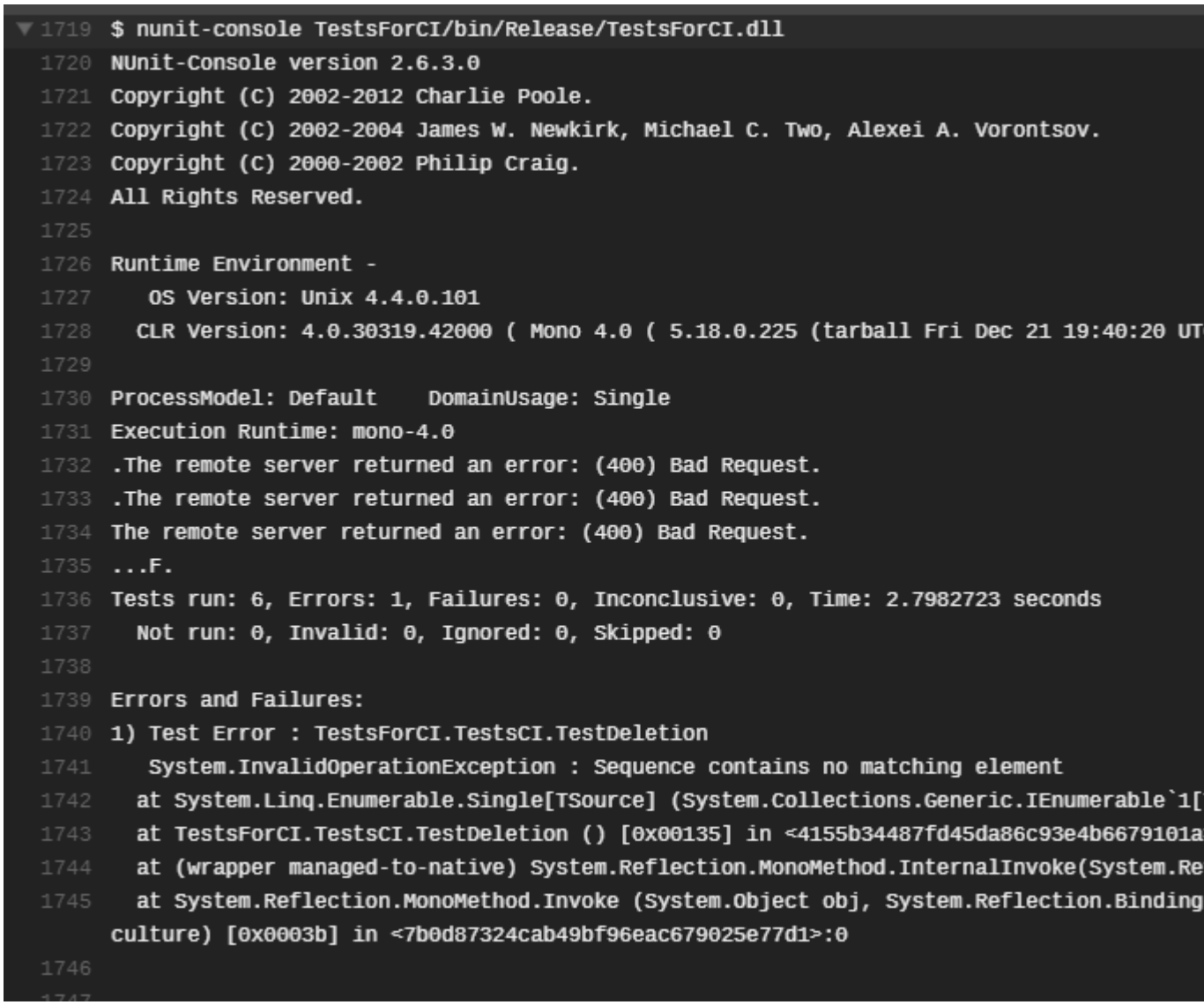

Рисунок 5.2 — Прохождение тестов после изменения метода DeleteGroup()

6 Функциональное тестирование

Функциональные тесты основываются на функциях, выполняемых системой. Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (use cases).

6.1 Use Cases

Use Case — это сценарная техника описания взаимодействия. С помощью Use Case может быть описано и пользовательское требование, и требование к взаимодействию систем, и описание взаимодействия людей и компаний в реальной жизни. В общем случае, с помощью Use Case может описываться взаимодействие двух или большего количества участников, имеющее конкретную цель. В разработке ПО эту технику часто применяют для проектирования и описания взаимодействия пользователя и системы, поэтому название Use Case часто воспринимает как синоним требования человека-пользователя к решению определенной задачи в системе.

Примеры Use Case для тестируемого приложения:

а) авторизация пользователя с вводом текущего города

б) добавление пользователем исполнителей в личный список подписок

в) удаление пользователем исполнителей из своего списка

г) вывод подписки в виде списка

д) вывод ближайших концертов исполнителей из подписок в городе пользователя

6.1.1 Основные действия пользователя

а) пользователь авторизует себя отправляя сообщение /start боту

б) пользователь указывает город через запрос /city 'город пользователя'. Не выполнив данный шаг, бот будет отказываться выполнять любые действия.

в) пользователь начинает заполнять свой лист подписок с помощью /add 'имя исполнителя'

г) пользователь выводит подписки с помощью команды /list

д) если пользователь хочет убрать исполнителя из подписок, то он вводит /remove 'имя исполнителя'

е) если же пользователь хочет полностью очистить лист подписок, то он вводит /clear

При всех запросах, указанных выше, сервер производит действия с базой данных для изменения/добавления информации.

6.1.2 Тестирование основных действий

Для проверки обработки основных запросов было проведено функциональное тестирование.

Таблица 6.1 — Функциональное тестирование

Обновление города пользователя	
Запрос	/city Town
Ожидаемый результат	"Ваш город обновлен, теперь вы находитесь в городе Town"
Добавление пользователя(начальная авторизация)	
Запрос	/city Town
Ожидаемый результат	"Поздравляем с регистрацией, ваш город Town"
Добавление пользователя с существующим userid	
Запрос	/city 'любой город'
Ожидаемый результат	Ничего
Примечание	данный тест необходим для сервера, т.к. в данном случае на сервер будет отправлена информация о возникшей ошибке. Пользователь же в данном случае никак не пострадает, т.к. если такой userid уже существует, то база данных никак не обновится. Информация серверу посылается для отладки, если в базе данных возникнут конфликты при добавлении пользователя с одинаковым первичным ключом.
Вывод списка подписок состоящей из 'group1'	
Запрос	/list
Ожидаемый результат	"1)group1 "
Удаление 3-ей группы из подписки с 3 группами	
Запрос	/remove group3
Ожидаемый результат	"Исполнитель group3 удален из подписок"
Удаление группы отсутствующей в подписках	
Запрос	/remove group22
Ожидаемый результат	Список подписок без изменений

Вывод пустого списка подписок	
Запрос Ожидаемый результат	/list "Добавьте группу с помощью команды ¨/add <Группа>:"
Попытка вывести список подписок, не указав город	
Запрос Ожидаемый результат	/list "Пользователь с таким id не найден. Добавьте город с помощью ¨/city"
Полная очистка подписок	
Запрос Ожидаемый результат	/clear "Ваш список подписок очищен"
Вывод списка подписок из 2-ух групп после удаление второй	
Запрос Ожидаемый результат	/remove group2 "1) group1"
Вывод списка подписок из 3-ух групп после удаление второй	
Запрос Ожидаемый результат	/remove group2 "1) group1 2)group3"
Вывод ближайших концертов(при отсутствии концертов в городе пользователя)	
Запрос Ожидаемый результат	/show "Не найдено ни одного концерта в вашем городе у исполнителей из вашего списка подписок, однако они выступают в России "
Вывод ближайших концертов(при наличии концерта в городе пользователя)	
Запрос Ожидаемый результат	/show "Инфо о концерте группы 0 Название: 1 Место: 2 Время: 3 Ссылка на источник: 4 "
Вывод ближайших концертов(при отсутствии концертов в принципе)	
Запрос Ожидаемый результат	/show "Не найдено ни одного концерта в вашем городе у исполнителей из вашего списка подписок "

6.2 Покрытие

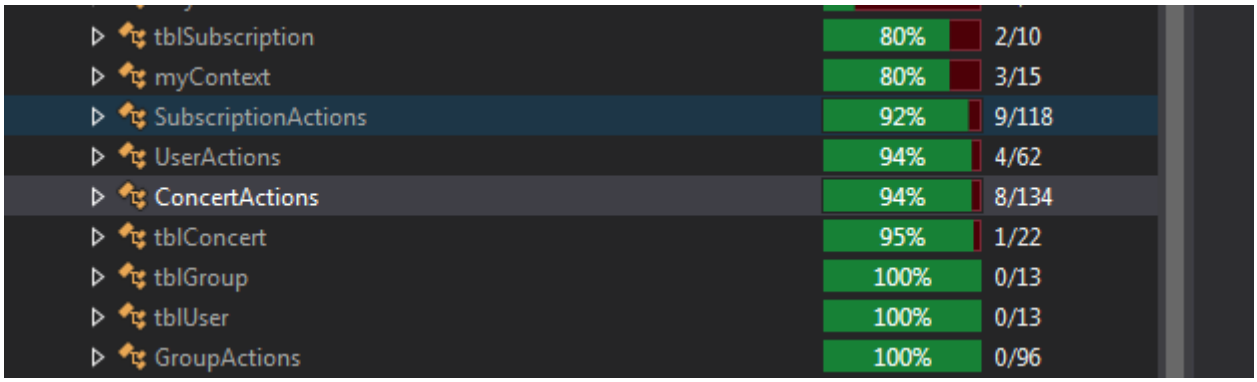

Рисунок 6.1 — Покрытие после модульных,интеграционных, функциональных тестов

Как можно заметить было достигнуто практически полное покрытие основных классов.