МИНИСТЕРСТВО ОБРАЗОВАНИЯ, КУЛЬТУРЫ И ИССЛЕДОВАНИЙ РЕСПУБЛИКИ МОЛДОВА

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ МОЛДОВЫ

ДЕПАРТАМЕНТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И АВТОМАТИКИ

Методическое пособие
для лабораторных работ
по дисциплине “ Тестирование
программных продуктов”
для студентов специальности TI, SI
(для дневного и заочного обучения)

Утверждено
Методической комиссией
факультета вычислительной техники,
информатики и микроэлектроники

Кишинёв 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ ... 3

ЛАБОРАТОРНАЯ РАБОТА № 1 ... 6

ЛАБОРАТОРНАЯ РАБОТА № 2 ... 18

ЛАБОРАТОРНАЯ РАБОТА № 3 ... 38

ЛАБОРАТОРНАЯ РАБОТА № 4 (1) ... 50

ЛАБОРАТОРНАЯ РАБОТА № 4 (2) ... 58

ЛАБОРАТОРНАЯ РАБОТА № 5 ... 68

ЛАБОРАТОРНАЯ РАБОТА № 6 ... 81

ЛАБОРАТОРНАЯ РАБОТА № 7 ... 99

ЛАБОРАТОРНАЯ РАБОТА № 8 ... 111

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ... 131

ПРИЛОЖЕНИЯ ... 133

Приложение А. Форма документа “Отчёт о проблеме” ... 133

Приложение Б. Информация по оформлению “Отчёта о проблеме” ... 134

Приложение В. Пример спецификации требований ... 139

Приложение Г. Пример составления отчета об ошибке ... 142

ВВЕДЕНИЕ

Цель данного курса заключается в изучении принципов и способах тестирования и верификации программных продуктов.

По окончанию курса студенты обретут: знания о принципах структурного и функционального тестирования, по вопросам организации процесса тестирования; способность составлять тестовые варианты для обнаружения ошибок и их документировать, анализировать и исправлять, и количественно оценивать комплексную сложность программных продуктов.

Основные понятия

Требования к программному обеспечению — совокупность утверждений относительно атрибутов, свойств или качества программной системы, подлежащей реализации.

Спецификация требований программного обеспечения (Software Requirements Specification) — законченное описание поведения системы, которую требуется разработать.

Дефект («баг») - ошибка в программе или системе, которая выдает неожиданный или неправильный результат относительно указанных в спецификации требований ПО.

Тестирование ПО - Процесс проверки соответствия заявленных к продукту требований и реально реализованной функциональности, осуществляемый путем наблюдения за его работой в искусственно созданных ситуациях и на ограниченном наборе тестов, выбранных определенным образом.

Тестирование по стратегии чёрного ящика – стратегия тестирования, при котором программа рассматривается как чёрный ящик (нет сведений о ее внутренней структуре, нет доступа к коду программы). Целью тестирования ставится выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

Тестировщик ПО - специалист, который проводит тестирование программного обеспечения, с целью обеспечения и контроля качества программного продукта или информационной системы.

Полный перечень пунктов, которые должны содержаться в спецификации требований программного обеспечения согласно стандартам

Введение:

a) цели;
b) соглашения о терминах;
c) предполагаемая аудитория;
d) масштаб проекта;
e) ссылки на источники.

Общее описание:

a) видение продукта;
b) функциональность продукта;
c) классы и характеристики пользователей;
d) среда функционирования продукта (операционная среда);
e) рамки, ограничения, правила и стандарты;
f) документация для пользователей;
g) допущения и зависимости.

Функциональность системы:

a) функциональный блок X (таких блоков может быть несколько);
b) описание и приоритет;
c) причинно-следственные связи, алгоритмы (движение процессов, workflows);
d) функциональные требования.

Требования к внешним интерфейсам:

a) интерфейсы пользователя (UI);
b) программные интерфейсы;
c) интерфейсы оборудования;
d) интерфейсы связи и коммуникации.

Нефункциональные требования:

a) требования к производительности;
b) требования к сохранности (данных);
c) критерии качества программного обеспечения;
d) требования к безопасности системы.

Прочие требования:

a) Приложение А: Глоссарий;
b) Приложение Б: Модели процессов и предметной области и другие диаграммы;
c) Приложение В: Список ключевых задач.

В практическом задании следует описать видение продукта/программы, его описание и назначение, а также функциональные требования. По возможности могут быть описаны остальные пункты. Спецификация требований должна быть описана в одном документе.

Пример спецификации требований для программы, реализующей сортировку вставкой, представлен в ПРИЛОЖЕНИИ В.

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: Основы тестирования и классификация ошибок. Составление спецификации требований. Интуитивное тестирование

Цель работы: *Тестирование программы, классификация ошибок и составление отчётов по ошибкам*.

Задание к лабораторной работе:

a) Написание программы и спецификации требований к ней в соответствии с вариантом задания к лабораторной работе. Каждое функциональное требование должно быть описано.

b) Написание тестовых сценариев по спецификации требований; тестирование функциональных требований разработанной программы: провести тестирование программы в соответствии с шагами тестирования, представленными в теоретической части и описание найденных дефектов/недочётов/ошибок.

Проект может быть написан на любом языке программирования, должен являться прикладной программой (реализованной с помощью интерфейса, input/output файлов, или связанной с базой данных), также это может быть сайтом и т. д.

Тестирование включает:

a) Написание тестовых сценариев для проверки функциональности.

b) Описание найденных ошибок в отчетах.

c) Исправление ошибок в программе и тестирование её вновь.

Содержание отчета:

1 Постановка задачи.

2 Составление спецификации требований.

3 Алгоритм программы.

4 Шаги тестирования.

5 Результаты работы программы.

6 Приведение классификации найденных ошибок. Составление отчёта на одну ошибку в соответствии с приложениями А и Б.

7 Приложение. Листинг программы.

Теоретическая часть

Пример серии тестов

Процесс тестирования программного обеспечения можно отчасти назвать интуитивным, но в то же время в основе его лежит вполне систематизированный подход. Хорошо протестировать программу означает нечто гораздо более серьезное, чем просто "погонять" ее несколько минут, чтобы убедиться, что она работает. Эффективное тестирование требует тщательного анализа и строгого системного подхода.

Эта лабораторная работа является своеобразным введением, цель которой на простом примере продемонстрировать подход к тестированию программ, применяемых опытными специалистами этого дела. Для примера была выбрана маленькая незамысловатая программка с несколькими ошибками.

Вместе с программой, которую следует протестировать, необходимо иметь на неё спецификацию (описание).

Пример спецификации программы

На вход программа принимает два параметра: x - число, n – степень. Результат вычисления выводится на консоль.

Значения числа и степени должны быть целыми.

Значения числа, возводимого в степень, должны лежать в диапазоне – [0..999].

Значения степени должны лежать в диапазоне – [1..100].

Если числа, подаваемые на вход, лежат за пределами указанных диапазонов, то должно выдаваться сообщение об ошибке.

Возьмём для демонстрации процесса тестирования программу, спецификация которой приведена ниже:

“*Назначение программы: сложить два введённых вами числа. В каждом из чисел должна быть одна или две цифры. Программа выполняет эхо-отображение вводимых чисел, а затем выводит их сумму. Ввод каждого числа завершается нажатием клавиши <Enter>. Запускается программа с помощью команды ADDER.*”

Первый цикл тестирования

Шаг 1. Начало тестирования с простого и наиболее очевидного теста.

Для начала с программой нужно познакомиться и посмотреть, достаточно ли она стабильна, чтобы её можно было тестировать. В программах, предоставленных для первого формального тестирования, часто сразу же происходит сбой. В первом тексте складываются числа 2 и 3. Последовательность действий и результаты приведены в таблице 1.1.

Таблица 1.1 – Порядок действия и результаты сложения чисел 2 и 3

№ действия	Ввод	Вывод
1	Вводите ADDER и нажимаете клавишу <*Enter*>	Экран мигает. Вверху экрана вы появляется знак вопроса.
2	Нажимаете 2	За знаком вопроса появляется цифра 2.
3	Нажимаете <*Enter*>	В следующей строке появляется знак вопроса.
4	Нажимаете 3	За вторым знаком вопроса появляется цифра 3.
5	Нажимаете <*Enter*>	В третьей строке появляется цифра 5. На несколько строк ниже появляется еще один знак вопроса.

На рисунке 1.1 видно, как выглядит экран по окончанию теста. Курсор указывает, где будет отображаться следующее вводимое число.

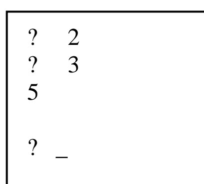


Рисунок 1.1 – Вид экрана по окончанию теста

Отчёт о проблемах, выявленных первым тестом:

“*Программа работает – она приняла числа 2 и 3 и вернула 5.*” Но проблемы, всё же, есть. Для их описания составляется отчёт, форма которого представлена в приложении А. Типы ошибок:

a) *Ошибка проектирования.* Нет никаких указаний на то, с какой программой работаете.
b) *Ошибка проектирования.* На экране нет никаких инструкций. Откуда знать, что нужно делать? Что, если вводите недопустимые числа? Отобразить инструкцию на экране не трудно, и она всегда будет перед глазами, в то время как печатная документация может потеряться.

c) *Ошибка проектирования.* Как остановить программу? Эта инструкция тоже должна быть на экране.

d) *Ошибка кодирования.* Сумма (число 5) выведена в стороне от слагаемых.

Примечание

Обязательно представляется отдельный “*Отчёт о проблеме*” (приложение А) по каждой ошибке. Описание всех четырех ошибок можно было бы поместить в один отчёт, но лучше этого не делать. Ошибки могут исправляться в разное время и сведения о тех из них, которые остались неисправленными, могут просто потеряться. Если программист захочет их сгруппировать, он сам рассортирует отчеты. Чтобы привлечь внимание к взаимосвязанным проблемам, просто поместите в отчеты соответствующие ссылки.

Шаг 2. Составление заметок о том, что ещё должно быть протестировано.

Выполнив первые, и самые очевидные тесты, следует подумать о том, что ещё следует протестировать. Свои соображения нужно записать: одни из записей примут форму заметок, другие же могут представлять собой достаточно строго формализованные описания серий тестов. Такие документированные группы тестов в дальнейшем могут послужить для проверки следующих версий программы. Примером может быть серия тестов, представленная в таблице 1.2, в которой для тестирования программы предлагается девять примеров. Тесты подобранны так, чтобы каждая цифра встречалась в них хотя бы один раз, чтобы было по одной комбинации чисел на каждую из вероятных проблем. А чтобы определить, на каких значениях вероятнее возникнут проблемы, эффективнее всего проверить граничные условия.

Вычисление количества возможных тестов не всегда является простой задачей. В допустимом диапазоне от -99 до 99 всего 199 чисел. Любое из них может стоять на первом месте и любое на втором. Всего получается $199^2 = 39601$ пар чисел. Это без учёта более сложных действий пользователя. Если же допустить использование клавиши редактирования, количество возможных тестов вырастет многократно. Задача определения количества возможных тестов относится к области математики (комбинаторный анализ). Из огромного количества возможных тестов нужно выбрать только наиболее важные тесты. Как только удаётся выделить группу (класс) однотипных тестов, можно провести несколько из них и проигнорировать остальные. Для отбора проводимых тестов есть важное правило: для выполнения всегда выбирайте из группы те тесты, на которых вероятнее всего ожидается сбой программы. Лучше всего подходят для тестирования примеры, лежащие на границе представленного классом диапазона значений. Именно на граничных значениях программы сбоят чаще всего.

Таблица 1.2 – Серия тестов для программы сложения двух чисел

Входные данные	Ожидаемый результат	Замечания
99 + 99	198	Пара наибольших чисел, которые может складывать программа.
-99 + -99	-198	В документации не сказано, что нельзя складывать отрицательные числа.
99 + -14	85	Большое первое число может повлиять на интерпретацию программой второго числа.
-38 + 99	61	Сложение отрицательного числа с положительным числом.
5 + 99	104	Проверка на то, влияет ли слишком большое второе число на интерпретацию первого.
9 + 9	18	9 является наибольшим числом из одной цифры.
0 + 0	0	Программы часто дают сбои при вводе нулей.
0 + 23	23	Программа может особым образом обрабатывать 0, поэтому его нужно проверить в виде первого слагаемого.
-78 + 0	78	Программа может особым образом обрабатывать 0, поэтому его нужно проверить в виде второго слагаемого.

Классом можно назвать группу значений, которые программа обрабатывает одним и тем же способом. Ограниченными значениями класса являются те входные данные, на которых программа меняет своё поведение. *Работа программиста* – проверять те практические точки, которые можно определить по листингу. *Задача тестировщика* – проанализировать программу с другой точки зрения, чтобы выявить те критические точки, которые программист пропустил. Поэтому классы возможных тестов следует выделять, исходя, прежде всего, из внешнего поведения программы. В результате набор тестов будет отличаться от того, который можно составить по листингу программы (именно в этом суть задачи тестировщика).

Важным моментом здесь является то, что границу значений обязательно нужно протестировать с двух сторон.

Шаг 3. Проверка допустимых значений и наблюдение за реакцией программы.

В таблице 2 приведены только допустимые значения входных данных программы. На следующем этапе тестирования можно создать такую же серию тестов для недопустимых значений. Ещё одна серия тестов может быть предназначена для проверки редактирования чисел: вводите значение, затем изменяете его и только после этого нажимаете <*Enter*>.

Шаг 4. Немного тестирования в режиме «свободного полёта».

Всегда записывайте, что делаете, и что происходит во время исследовательских тестов.

От формальных тестов следует перейти к неформальным тестам. В таблице 1.3 видно, что при малейшей провокации программа работает неверно (она “зависает”). Это приводит к затрате большего времени на перезапуск компьютера, нежели чем на тестирование.

Таблица 1.3 – Неформальные тесты для программы сложения двух чисел

Тест	Особенности теста	Замечания
100 + 100	Граничное условие – числа больше максимального допустимого значения (99).	Программа приняла 10. Когда ввели второй 0, чтобы получилось 100, программа повела себя так, как будто вы нажали <*Enter* >. Так же было и со вторым числом 100. В результате по окончании теста на экране было следующее: ? 10 ? 10 20
<*Enter*> + <*Enter*>	Проверка реакции программы на отсутствие числовых данных.	Когда нажали <*Enter*>, программа напечатала 10 – последнее введенное вами число. То же было и после второго нажатия <*Enter*>, и в качестве суммы программа напечатала 20.
123456 + 0	Ввод более длинных чисел.	Программа приняла первые две цифры и проигнорировала остальные так же, как было с числом 100. В будущей версии программа будет принимать большие числа.
1,2 + 5	Числа с десятичной частью.	Реакция на десятичный разделитель такая же как и на <*Enter*>.
A + b	Недопустимые символы.	Когда нажали <*Enter*> после <*A*> программа “зависла”. Для продолжения тестирования приходится перезапускать компьютер.
<*Ctrl + A*>, <*Ctrl + B*>, <*Ctrl + C*>, <*Ctrl + D*>, <*F1*>, <*Esc*>	Управляющие символы и функциональные клавиши часто являются источниками проблем.	Для всех комбинаций клавиш, кроме <*Ctrl + C*>, программа отобразила графические символы, затем, после нажатия <Enter>, “зависла”. Нажатие <*Ctrl + C*> привело к завершению программы и выходу в операционную систему.

Столкнувшись с очередной проблемой, составляете о ней отчет. О сданных отчётах лучше написать для себя итоговые заметки. На этом тестирование первой версии программы можно считать завершенным.

Шаг 5. Подведение итогов о программе и выясненных недостатках.

Эта работа не всегда необходима, но часто оказывается очень полезной. До этого времени все время были сконцентрированы на конкретных деталях – анализировали допустимые данные, продумывали граничные условия и составляли тестовые примеры. В будущем больше времени будет тратиться на выполнение уже подготовленных тестов, чем на придумывание новых. Сейчас же самое время мысленно отступить немного назад и окинуть взглядом программу в целом, увидеть ее недостатки и продумать стратегию будущего тестирования.

Итоги первого цикла тестирования

Начали с простейшего из возможных тестов. Поскольку программа его прошла, разработали серию формальных тестов, чтобы проверить, как она работает с допустимыми данными. Эти тесты будут использованы и дальше. Поскольку часть проверок программа не прошла, на планирование дальнейших серий тестов времени можно не тратить. Вместо этого проведём несколько неформальных экспериментов и выясним, что программа вообще очень нестабильна. Записали несколько замечаний, к которым обратимся при тестировании следующей версии программы.

Если бы программа успешно прошла первую серию тестов, разрабатывается вторая, более обстоятельная. Если бы программа снова показала себя надежной, продолжается ее тестирование, пока не исчерпались бы идеи или отведенное время. Напоследок провели бы несколько беглых тестов, не входивших в ранее разработанные серии, и записали замечания на будущее. Завершив тестирование и всю бумажную работу, стоит потратить еще немного времени, чтобы обдумать результаты. Выполнили самое очевидное, но это еще только начало. Пока нет конкретного плана. Просто делали то, что первым приходило в голову. По ходу работы были определены две линии атаки. Теперь же *нужно выделить время* на обдумывание. Это очень важно, даже если сроки сильно поджимают.

Второй цикл тестирования

Поговорили с программистом, и он сказал, что скорость работы программы исключительно важна, а вот объём кода не имеет никакого значения. Резолюции программиста на отчётах тестировщика:

a) Ошибка проектирования: на экране нет названия программы.
   Резолюция: не будет исправлена.
b) Ошибка проектирования: на экране нет инструкций.
   Резолюция: не будет исправлена.
   Примечание: замечание верное, но вывод инструкций замедлит работу программы.
c) Ошибка проектирования: невозможность остановить программу.

Резолюция: исправлена; на экране отображается подсказка – “*Для выхода нажмите <Ctrl + C>*”.

d) Ошибка кодирования: сумма (5) выводится в стороне от слагаемых.

Резолюция: исправлена.

e) Ошибка кодирования: программа “зависает” на отрицательных числах.

Резолюция: исправлена; программа будет складывать и отрицательные числа.

f) Ошибка кодирования: программа интерпретирует третий введенный символ как нажатие *<Enter>*.

Резолюция: в работе (ещё не исправлена).

g) Ошибка кодирования: сбой при вводе нечисловых данных.

Резолюция: не проблема.

Примечание: не выполняйте этой операции.

h) Ошибка кодирования: сбой при вводе управляющих символов.

Резолюция: не проблема.

Примечание: не используйте управляющие символы.

i) Ошибка кодирования: сбой при нажатии функциональных клавиш.

Резолюция: не проблема.

Примечание: не используйте функциональные клавиши.

Шаг 1. Ознакомление с резолюцией программиста.

Так ознакомившись с резолюцией программиста, узнаете, что нужно делать, а чего не нужно. Как видите, хорошо, что не разрабатывали тестов для проверки кода обработки ошибок: этого кода нет и не будет. Более того, хотя программа и будет теперь обрабатывать отрицательные числа, она будет обрабатывать их не все, а только до -9. Числа от -10 до 199 длиной в три символа она по-прежнему не обрабатывает, интерпретируя третий символ как нажатие клавиши *<Enter>*. Обратившись к разработанной ранее серии тестов из таблицы 6, видно, что тесты с числами -99, -78 и -14 запускать не придётся. Вместо -78 и -14 необходимо взять пару однозначных отрицательных чисел.

Очень часто программист просит протестировать остальную часть программы, в то время как он занимается исправлением уже найденных ошибок – и это разумно. Некоторые из запланированных тестов нет смысла проводить до исправления ошибки, с другими же вполне можно поработать. Если ждать, пока можно будет провести “лучшие” из тестов, можно оставить без внимания целые области программы, на которые потом уже не хватит времени. В нашем примере можно протестировать числа между -1 и -9 – так хоть и не полностью, но будет проверено, как работает сложение отрицательных чисел, вместо того чтобы опустить эту проверку вообще.

Из резолюций на отчетах видно, какие тесты больше проводить не нужно, а какие нужно заменить новыми.

Шаг 2. Анализ комментариев к ошибкам, которые не будут исправлены.

Возможно, следует провести дополнительное тестирование. В нашей программе хуже всего обстоит дело с обработкой ошибок. Программист не собирается исправлять ситуацию. Как же быть? Чтобы добиться исправления ошибки, нужно продемонстрировать ситуацию, в которой её появление абсолютно недопустимо.

Чтобы придумать самые показательные примеры недопустимого поведения программы, необходимо постараться выявить суть ситуации. В нашем случае программа “зависает”, когда нажимаются определенные клавиши. Так она ведет себя с буквами, управляющими и функциональными клавишами. Фактически программа “зависает” при вводе любого недопустимого (нецифрового) символа. Программист говорит, что таких символов вводить не должны. С точки зрения тестировщика, программа должна вести себя вежливо и не заставлять перезапускать компьютер каждый раз, когда будет сделано что-то не так.

Программа также некорректно ведет себя в ответ на нажатие некоторых клавиш. Программист считает, что это не страшно, поскольку никто и не ждет, что программа примет эти клавиши. А что, если программа “зависнет” в ответ на ввод символов, которые, по мнению пользователя, она должна принять?! Если найти достаточно таких символов, программисту придется написать код для их обработки, а заодно уж он может обработать и остальные символы. Подумаем, какие же клавиши люди могут нажимать при работе с арифметической программой. Здесь нужно применять методику *мозгового штурма.* Запишите все клавиши, которые могут прийти человеку. Запишите, почему эти клавиши могут показаться уместными, при этом беспокойтесь, согласится ли программист с вашими предположениями – позднее можно ещё пересмотреть список и отобрать из него самое существенное. Вот вероятный список таких клавиш:

a) цифры;
b) знак минус (‘-’);
c) знак плюс (‘+’);
d) пробел до числа;
e) пробел после числа;
f) операции умножения и деления без остатка (‘*’, ‘/’);
g) знак доллара (‘$’);
h) знак процента (‘%’);
i) скобки;
j) клавиша <*Backspace*>;
k) клавиша <*Detete*>;

l) клавиша <*lnsert*>;

m) клавиши управления курсором.

Шаг 3. Просмотр записей, которые были сделаны в прошлый раз.

Необходимо добавить к предыдущим замечаниям новые и приступить к тестированию. Сначала повторяются старые тесты и убедившись, что программа по-прежнему сможет сложить 2 и 2 и не получить при этом 5, т. е. обязательно тестируется логика программы.

По характеру дела видно, что программа отображает подсказку "*Для выхода нажмите «Ctrl + C»*" по каждой операции сложения. Вот что отображается при этом на экране:

? 99

? 99

 198 – Для выхода нажмите «Ctrl + C»

? -9

? -9

 -18 – Для выхода нажмите «Ctrl + C»

? _

Программист говорит, что программа должна работать быстро. В таком случае всё, на что тратится лишнее время, является ошибкой.

Составляете отчёты по следующим ошибкам:

a) *Ошибка проектирования*. Для вывода на экран подсказки тратится лишнее компьютерное время. Поскольку одной из задач разработки является создание очень быстрой программы, это ошибка. Почему бы просто не написать "*Для выхода нажмите «Ctrl + C»"* внизу экрана, сразу при запуске программы и никогда больше ничего не выводить в этой строчке? Если программист достаточно аккуратен, большая часть тестов, включая и те, которые потребовали наибольшей изобретательности, не выявит ошибок.

b) *Ошибка кодирования*. Проблема такова – в ответ на нажатие клавиш редактирования и других предположительно допустимых клавиш программа "зависает". Предлагаемое исправление – проверять каждый вводимый символ. Недопустимые символы игнорировать или выводить сообщение об ошибке.

Хороший тестировщик – не тот, кто выявит больше всего ошибок, и не тот, кто заставит смутиться даже самого первоклассного программиста. Лучшим является тот, кто добьётся исправления наибольшего количества ошибок.

Варианты задания к лабораторной работе

1. Нахождение наибольшего общего делителя при условии, что начальные значения x_1 и x_2 положительны.
2. Вычисление факториала неотрицательного целого числа.
3. Вычисление чисел Фибоначчи.
4. Поиск заданного символа в строке.
5. Выполнение конкатенации (сцепления) строк.
6. Определить частное q и остаток r от деления x на y.
7. Линейный поиск в упорядоченном по возрастанию элементов массиве $A[1..n]$. Определить порядковый номер m некоторого значения x.
8. Поиск значения x в двумерном массиве.
9. Написать программу, которая по данному фиксированному массиву $B[0..n-1]$, где $n > 0$, записывает в d число нечётных значений в $B[0..n-1]$.
10. Нахождение наибольшего целого числа по данному фиксированному целому числу m > 0, которое не больше n, в массиве целых чисел.
11. Суммирование элементов массива $B[0..n-1]$, где $n > 0$.
12. Вставка пробелов (выравнивание строки текста по правому краю).
13. Выравнивание текста по центру.
14. Определение максимальной возрастающей подпоследовательности значений, идущих не обязательно подряд.
15. Перестановка двух неперекрывающихся сегментов массива равного размера.
16. Подсчёт числа листьев в дереве.
17. Удаление лишних пробелов в тексте.
18. Суммирование комплексных чисел.
19. Умножение комплексных чисел.
20. Деление комплексных чисел.
21. Выполнение арифметических операций над комплексными числами.
22. Вводится строка слов. Вывести слова в обратном порядке.
23. В одномерном массиве найти максимальный из отрицательных элементов и поменять его местами с последним элементом массива.
24. В зависимости от выбора пользователя программа производит либо шифрование, либо дешифровку вводимой строки методом кода (шифра) Цезаря - каждая буква заменяется на букву, отстоящую в алфавите от исходной на определенное значение.
25. Шифр Виженера.
26. Генератор случайных чисел.

27. Составьте программу печати прямоугольного треугольника из звездочек

```
*
**
***
****
*****
```

используя цикл for. Введите переменную, значением которой является размер катета треугольника.

28. Напишите программу с циклами, которая рисует треугольник:

```
    *
   ***
  *****
 *******
*********
```

29. Напишите программу, подсчитывающую число символов, поступающих со стандартного ввода.

30. Составьте программу перекодировки вводимых символов со стандартного ввода по следующему правилу:

   a -> b

   b -> c

   c -> d

   ...

   z -> a

   другой символ -> *

   Коды строчных латинских букв расположены подряд по возрастанию.

31. Напишите программу, печатающую квадраты и кубы целых чисел.
32. Напишите программу, печатающую сумму квадратов первых n целых чисел.
33. Напишите программу, которая переводит секунды в дни, часы, минуты и секунды.
34. Напишите программу, переводящую скорость из километров в час в метры в секундах.
35. Напишите программу, шифрующую текст файла путем замены значения символа (например, значение символа **C** заменяется на **C+1** или на **~C**).

Контрольные вопросы

1. Что вы понимаете под тестированием и верификацией программных продуктов?
2. Какие типы ошибок существуют?
3. Что такое классы тестов и для чего они предназначены?
4. На какие шаги можно разбить первый цикл тестирования любой программы?
5. Для чего нужен второй цикл тестирования?

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: Тестирование «белым ящиком». Структурное тестирование.

Цель работы: Тестирование программного продукта с помощью следующих способов тестирования «белым ящиком»:

a) Тестирование базового пути.
b) Тестирование методом ветвей и операторов отношений.
c) Тестирование потоков данных.
d) Тестирование циклов.

Содержание отчета:

1 Постановка задачи.
2 Алгоритм программы на псевдокоде.
3 Пояснение шагов способа тестирования базового пути.
4 Потоковый граф.
5 Вычисление цикломатической сложности.
6 Пояснение шагов способа тестирования ветвей и операторов отношений.
7 Пояснение способа тестирования потоков данных.
8 Граф программы с управляющими и информационными связями.
9 Шаги тестирования циклов.
10 Тестовые варианты, полученные по каждому методу тестирования, приводятся после рассмотренного метода тестирования.
11 Описание реализации способов тестирования на ЭВМ:
- скриншот вывода на экран тестовых вариантов и реальных результатов;
- скриншот вывода на экран ожидаемого и полученного путей тестирования для каждого тестового варианта.

11 Приложение. Листинг программы.

Теоретическая часть

Стратегия «белого ящика», или стратегия тестирования, управляемого логикой программы, позволяет исследовать внутреннюю структуру программы. При данном тестировании программа рассматривается как объект с известной внутренней структурой. Поэтому такой принцип тестирования называется структурным, или «стеклянным ящиком», или «белым ящиком». Тестировщик (как правило, программист) разрабатывает тесты, основываясь на знании исходного кода, к которому он имеет полный доступ. Главной его идей является правильный выбор

тестируемого программного пути. Таким образом, тестирование методом «белого ящика» обладает следующими *преимуществами:*

a) *Направленность тестирования*. Программист может тестировать программу по частям, разрабатывать специальные тестовые подпрограммы, вызывающие тестируемый модуль, и передают ему интересующие данные. Отдельный модуль гораздо легче протестировать именно, когда он является открытым.

b) *Полный охват кода*. Тестировщик всегда может определить, какие именно фрагменты кода работают в каждом тесте. Он может увидеть какие ветви кода остались не протестированными и может подобрать условия, в которых они будут выполнены.

c) *Управление потоком*. Программист знает, какая функция должна выполняться в программе, следующей и каким должно быть её текущее состояние. Чтобы выяснить, правильно ли работает программа, программист может включить в неё отладочные команды, отображающие информацию о ходе выполнения программы.

d) *Отслеживание целостных данных*. Программисту известно, какая часть программы должна изменять каждый элемент данных. Отслеживая состояния данных (к примеру, с помощью отладчика), можно выявить такие ошибки, как изменение данных не теми модулями, их неверная интерпретация или плохая организация.

e) *Внутренние граничные точки*. В исходном коде видны те граничные точки программы, которые скрыты при отсутствии такого кода. К примеру, для выполнения некоторого действия можно использовать множество различных алгоритмов, и, не заглянув в код, невозможно определить, какой из этих алгоритмов выбрал программист. Другим типичным примером является проблема переполнения буфера для хранения входных данных. Программист сразу может сказать, при каком количестве данных буфер переполнится, и ему при этом не нужно проводить тысячи тестов.

f) *Тестирование, определяемое выбранным алгоритмом*. Для тестирования обработки данных, использующей очень сложные вычислительные алгоритмы, могут понадобиться специальные технологии. В качестве классического примера можно привести сортировку данных. Тестировщику нужно знать точно, какие алгоритмы используются, и обратиться к специальной литературе.

Тестирование «белого ящика» можно рассматривать как часть процесса программирования. Программисты регулярно выполняют это тестирование, после написания программы, её модулей и даже отдельных блоков кода.

Существуют четыре стратегии (способа) тестирования методом «белого ящика»:

a) Тестирование базового пути (покрытие решений).

b) Тестирование условий (покрытие условий).

c) Тестирование потоков данных.

d) Тестирование циклов.

Тестирование по принципу “белого ящика” характеризуется степенью, в какой тесты выполняют или покрывают логику (исходный текст) программы. Исчерпывающее тестирование по принципу белого ящика предполагает выполнение каждого пути в программе, но поскольку в программе с циклами выполнение каждого пути обычно нереализуемо, то тестирование всех путей не рассматривается.

Остановимся на рассмотрение каждого из этих способов тестирования “белого ящика”.

Тестирование базового пути

Базовые понятия

Первым будет рассмотрен способ тестирования базового пути, который был разработан в 1976 году Томом Мак-Клером. К главным преимуществам этого способа относятся:

a) возможность получить оценку комплексной сложности программы;

b) возможность определить необходимое количество тестовых вариантов на основании данной оценки.

Для усвоения рассматриваемого способа тестирования следует изложить и пояснить ряд базовых понятий. Написанная программа представляется в виде потокового графа. К особенностям такого графа можно отнести:

a) граф строится отображением управляющей структуры программы;

b) вершины (узлы) графа соответствую линейным участкам программы;

c) граф ориентирован и его дуги отображают поток управления в программе;

d) среди вершин различают операторные вершины и предикатные вершины;

e) предикатные вершины соответствуют простым условиям. Составное условие (условие, содержащее одну или несколько булевых операций) отображается через набор предикатных вершин;

f) замкнутые области на графе, образованные дугами и вершинами, называются регионами. При этом, окружающая граф среда, рассматривается как дополнительный регион.

Согласно данной стратегии тестирования должно быть записано достаточное число тестов, такое, что каждое решение на этих тестах примет значение истина и ложь, по крайней мере, один раз. Иными словами, каждое направление перехода должно быть реализовано хотя бы один раз. Примерами операторов перехода или решений являются операторы *while* или *if*.

Цикломатическая сложность

Цикломатическая сложность представляет собой метрику программного обеспечения, которая обеспечивает количественную оценку логической сложности программы. Цикломатическая

сложность определяет количество независимых путей в базовом множестве программы, а также определяет верхнюю оценку количества тестов.

Все независимые пути графа образуют базовое множество. К свойствам базового множества относятся:

a) мощность базового множества равна цикломатической сложности потокового графа;

b) в случае выполнения некоторых тестов гарантируется проверка базового множества.

Тестовый вариант — это однократное выполнение каждого оператора и проверка выполнения каждого условия по ветви FALSE и по ветви TRUE.

Существует три способа для вычисления цикломатической сложности:

a) посредством определения количества регионов: $V(G) = R$, где R - количество регионов;

b) по формуле через количество вершин и дуг: $V(G) = E - N + 2$, где E - количество дуг, N - количество вершин;

c) по формуле через предикатные узлы: $V(G) = P + 1$, где P – количество предикатных узлов.

Алгоритм тестирования методом базового пути

Тестирование данным методом будет рассмотрено на примере процедуры сортировки вставкой. Алгоритм сортировки вставкой в псевдокоде записывается следующим образом:

```
Insertion_Sort(A)
1 for j ← 2 to length[A]
2     do key ← A[j]
3          i ← j - 1
4                while i > 0 и A[i] > key
5                 do A[i + 1] ← A[i]
6                      i ← i - 1
7                A[i + 1] ← key
8 end for
```

Шаги тестирования методом базового пути:

Шаг 1. Построение потокового графа на основе выше представленного текста программы. Для начала следует пронумеровать операторы текста, разбив его тем самым на логические блоки.

Изменение нумерации операторов в соответствие с логикой программы:

```
1 for j ← 2 to length[A]
2     do key ← A[j]
2          i ← j - 1
3 и 4        while i > 0 и A[i] > key
5                do A[i + 1] ← A[i]
5                i ← i - 1
6            end loop
7          A[i + 1] ← key
```

```
7 end for
8 end Sort
```

Пусть length[A] = n.

На построенном графе также обозначаются регионы (рисунок 2.1).

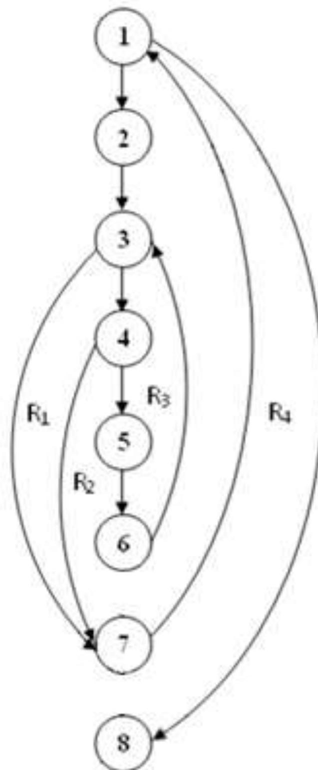


Рисунок 2.1 – Потоковый граф процедуры Insertion_Sort

Шаг 2. Определение цикломатической сложности построенного потокового графа. Для этого используются все три вышеописанных способа:

a) $V(G) = 4$;

b) $V(G) = 10 - 8 + 2 = 4$;

c) $V(G) = 3 + 1 = 4$.

Таким образом, цикломатическая сложность потокового графа равна 4.

Шаг 3. Определение базового множества независимых путей, количество которых определяется по цикломатической сложности:

Путь 1: 1-8 // n =1, массив состоит из одного элемента;

Путь 2: 1-2-3-7-1-8 // $n > 1$, является частью 4-го пути;

Путь 3: 1-2-3-4-7-1-8 // $n > 1$, массив отсортирован;

Путь 4: 1-2-3-4-5-6-3-…-7-1-8 // $n > 1$ режим нормальной обработки.

Шаг 4. Подготовка тестовых вариантов (ТВ):

ТВ 1: Исходные данные: $n = 1$, $A = \{5\}$ – массив состоит из одного элемента.

Ожидаемый результат: $A = \{5\}$; массив упорядочен.

ТВ 2: Исходные данные: $n = 5$, $A = \{5, 10, 1, 18, 6\}$ – не отсортированный массив от 2 до n элементов.

Ожидаемый результат: $A = \{1, 5, 6, 10, 18\}$; массив упорядочен.

ТВ 3: Исходные данные: $n = 5$, $A = \{5, 10, 1, 18, 6\}$ – проверка условия остановки сдвижки элементов массива (размером от 2 до n) влево ($i > 0$).

Ожидаемый результат: $A = \{1, 5, 6, 10, 18\}$; массив упорядочен.

ТВ 4: Исходные данные: $n = 4$, $A = \{99, 78, 2, 41\}$ – массив из i, где $i \geq 3$.

Ожидаемый результат: $A = \{2, 41, 78, 99\}$; массив упорядочен.

Тестирование условий

Базовые понятия

Различают три типа условий:

a) простое условие:
   1) булева переменная;
   2) выражение отношения (арифметическое выражение)

   Вид:

   $E_1 <$ *оператор отношения* $> E_2$,

   где $<$ *оператор отношения* $>= \{<, >, =, \geq, \leq, \neq\}$

b) составное условие. Оно содержит в себе несколько простых условий и включает:
   1) простые условия;
   2) булевы операторы (OR, AND, NOT);
   3) пара скобок.

в) булево выражение, не содержащее в себе арифметических выражений.

Существуют методики тестирования условий – это *методика тестирования ветвей* и *методика тестирования области определения*.

Тестирование ветвей – это простейшая методика, позволяющая проверять простые условия и ветви TRUE / FALSE. Вторая методика – *методика тестирования области определения* – тестирует выражения отношения. Составляются следующие тесты для логических выражений:

$E_1 < E_2$;

$E_1 > E_2$;

$E_1 = E_2$.

Тестирование условий имеет следующие достоинства:

a) достаточно простое выполнение измерения тестового покрытия условия;

b) тестовое покрытие условий в программе является фундаментом для генерации дополнительных тестов программы;

c) данная методика эффективна не только для обнаружения ошибок в условиях, но также и для обнаружения других ошибок в программе.

Методика *тестирования ветвей и операторов отношений* была разработана в 1989 году. Она применима при выполнении следующих ограничений:

a) все булевы переменные и операторы отношения входят в условие только по одному разу;
b) в условии отсутствуют общие переменные.

Ограничение на результат фиксирует возможные значения аргумента простого условия, если он один, или соотношение между значениями аргументов, если их несколько. Ограничение составного условия обозначается следующим образом:

$ОУ_C = (d_1, d_2, \dots, d_i, \dots, d_n)$,

где n – количество простых условий в составном условии.

Если, к примеру, d_1 – булевы переменные, то $d_1 = (1{,}0)$. Если d_1 – выражение отношения, то $d_2 = (>, =, <)$.

На основе ограничения составного условия ($ОУ_C$) строится *ограничивающее множество ОМ*. Элементами ограничивающего множества являются сочетания всех возможных значений. Построение ОМ выполняется путем подстановки в константные формулы $ОМ_{\&}$ или $ОМ_{OR}$. Например,

для условия типа И *(a & b)*

$ОМ_{\&}$ = {(false, true), (true, false), (true, true)};

для условия типа ИЛИ *(a от b)*

OMor = {(false, false), (false, true), (true, false)}.

Алгоритм тестирования методом ветвей и операторов отношений

Если отказаться полностью от тестирования всех путей, то можно показать, что критерием покрытия является выполнение каждого оператора программы, по крайней мере, один раз. Это метод покрытия операторов отношений. К сожалению, это слабый критерий, так как выполнение каждого оператора, по крайней мере, один раз есть необходимое, но недостаточное условие для приемлемого тестирования по принципу белого ящика.

Метод тестирования ветвей и операторов отношений выполняется по следующему алгоритму:

a) построение множества ограничений условий;
b) выявление ограничения результата по каждому простому условию;
c) построение ограничивающего множества, посредством подстановки значений в константные формулы;
d) для каждого элемента ограничивающего множества разрабатываются тестовые варианты.

Тестирование этим методом будет также рассмотрено на примере процедуры сортировки вставкой. В данной процедуре имеет место составное условие (строка 4 листинга процедуры).

Insertion_Sort (*A*)

```
1 for j ← 2 to length[A]
2      do key ← A[j]
2           i ← j - 1
3 и 4        while i > 0 и A[i] > key
5                 do A[i + 1] ← A[i]
5                 i ← i - 1
6            end loop
7           A[i + 1] ← key
7 end for
8 end Sort
```

Таким образом, имеется выражение ($i > 0$) & ($A[i] > key$). Это выражение является составным и содержит в себе два простых выражения.

Множество ограничения составного условий: $ОУ_С = (d_1,\ d_2)$, где $d_1 = (\le, >)$; $d_2 = (\le, >)$.

Ограничение на результат: $OP = (>, >)$.

Константная формула:

$C_{\&} = a \;\&\; b$,

где $a \leftrightarrow (i > 0)$,

$b \leftrightarrow (A[i] > key)$.

Таблица значений формулы для построения ограничивающего множества (таблица 2.1) представлена ниже.

Таблица 2.1 – Значения переменных для отношения выражения

a	b	&
0	0	0
0	1	0
1	0	0
1	1	1

Ограничивающее множество: $OM_{\&} = \{(0,1),(1,0),(1,1)\} = \{(\le, >),(>, \le),(>, >)\}$.

Построение тестовых вариантов:

ТВ 1: Исходные данные: $n = 5$, $A = \{3, 2, 8, 9, 1\}$ – проверка остановки сдвижки.

Ожидаемый результат: $A = \{3, 2, 8, 9, 1\}$.

ТВ 2: Исходные данные: $n = 9$, $A = \{9, 8, 17, 298, 1, 1, 4, 2, 4\}$ – проверка на то, нарушает ли текущий элемент (*key*) порядок в массиве размером от 2 до n элементов.

Ожидаемый результат: $A = \{1, 1, 2, 4, 4, 8, 9, 17, 298\}$.

ТВ 3: Исходные данные: $n = 3$, $A = \{10, 2, 8\}$ – проверка процесса сортировки.

Ожидаемый результат: $A = \{2, 8, 10\}$.

Тестирование потоков данных

Базовые понятия

Тестирование потоков данных – это имя, которым называют группу тестовых стратегий, основанную на выборе путей, таким образом, чтобы отследить изменение данных.

По крайней мере, половина современных исходных кодов состоят из операторов объявления данных. Если хотим достичь качественного кода, который можно будет часто повторно применять, то нам обязательно стоит обратиться к данному методу тестирования. Вас это не убедило? В современных ПК в компромиссе «память – процессорное время» в 99% случаев выбор делают в пользу процессора. А алгоритмы, предполагающие работу с большими массивами данных, обязательно нужно тестировать на корректность потоков данных.

*Ошибки (*баги*), выявляемые таким типом тестирования.* Типовые баги, обнаруживаемые при таком типе тестирования – это баги, связанные с ограниченной областью видимости данных, баги определения и удаления данных, перезаписи данных в переменную, содержавшую и до этого полезную информацию и так далее. Зачастую подавляющее большинство ошибок, выявляемых при тестировании базового пути, могут быть найдены этим способом тестирования.

Граф потоков данных – это граф, состоящий из узлов и направленных связей (то есть, связей со стрелками на их концах). Данные могут быть **определены** либо **использованы.**

Определение – Переменная определяется в двух случаях:

a) при объявлении, при этом получая значения по умолчанию;
b) при появлении в левой части выражения присваивания.

Использование – Переменная используется в двух случаях:

a) при появлении в правой части выражения присваивания;
b) в случае появления в условиях *if*, условиях, задающих цикл (*while*, *for*), то есть в случаях управления ходом выполнения программы.

Шаги тестирования потоков данных (DU тестирование)

Способ DU тестирования требует охвата всех DU цепочек программы. Разработка тестов проходит на основании потоков данных. Критерием для выбора пути является покрытие максимального количества DU цепочек.

Шаги способа DU тестирования:

Шаг 1. Построение управляющего графа программы (описано в методе тестирования базового пути).

Шаг 2. Построение информационного графа. Граф строится по следующим принципам:

a) в случае определения переменной создаётся направленная связь (пунктирная линия) от вершины к переменной;

b) в случае использования переменной создаётся направленная связь (пунктирная линия) от переменной к вершине (в противоположную сторону).

Шаг 3. Формирование полного набора DU цепочек.

Шаг 3а. Формирование множества определения данных:

$DEF(i) = \{x \mid i - ая\ \ вершина\ \ содержит\ \ опредение\ \ x\}$

Шаг 3б. Формирование множества использования данных:

$USE(j) = \{x \mid j - ая\ \ вершина\ \ использует\ \ x\}$

Шаг 3в. Формирование цепочек определения и использования:

DU представляется в виде $[x, i, j]$, где x – переменная, i – вершина определения x, j – вершина использования x.

Шаг 4. Формирование полного набора отрезков путей в управляющем графе (на основе построенных DU цепочек).

Шаг 5. Построение маршрутов.

Шаг 6. Формирование тестовых вариантов.

К достоинствам метода относят его относительную простоту анализа и автоматизации. В то же время трудности в выборе минимального количества эффективных тестов осложняют его использование.

Тестирование потоков данных можно провести, к примеру, на алгоритме сортировки вставкой (Insertion Sort), который эффективно работает при сортировке небольшого количества элементов. Приведём формулировку задачи:

Вход: последовательность из n чисел $< a_1, a_2, ..., a_n >$.

Выход: перестановка (изменение порядка) $< a'_1, a'_2, ..., a'_n >$ входной последовательности таким образом, что для её членов выполняется соотношение $a'_1, \le a'_2 \le ... \le a'_n$.

Ниже приведён псевдокод алгоритма, который и будем тестировать:

```
0 // точка вызова процедуры сортировки – начальная (внешняя) вершина
1 for j ← 2 to length[A]
2      do key ← A[j]
2           i ← j - 1
3 и 4        while i > 0 и A[i] > key
5                  do A[i + 1] ← A[i]
5                  i ← i - 1
6            end loop
7           A[i + 1] ← key
7 end for
8 end Sort
```

Шаг 1. Построение управляющего графа программы (рисунок 2.2).

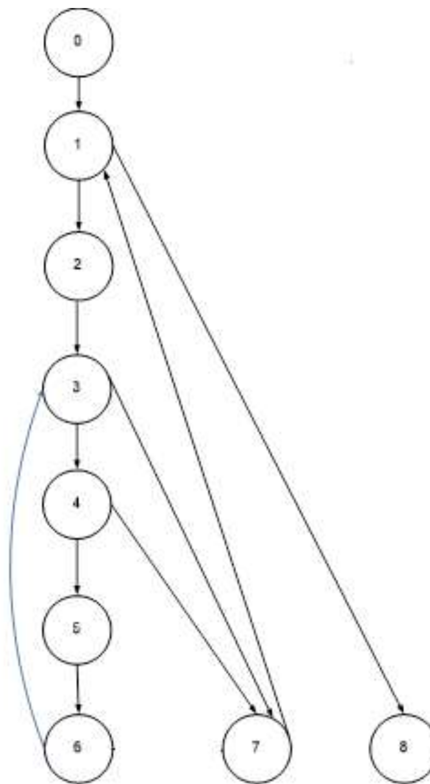


Рисунок 2.2 – Управляющий граф программы

Шаг 2. Построение информационного графа (рисунок 2.3).

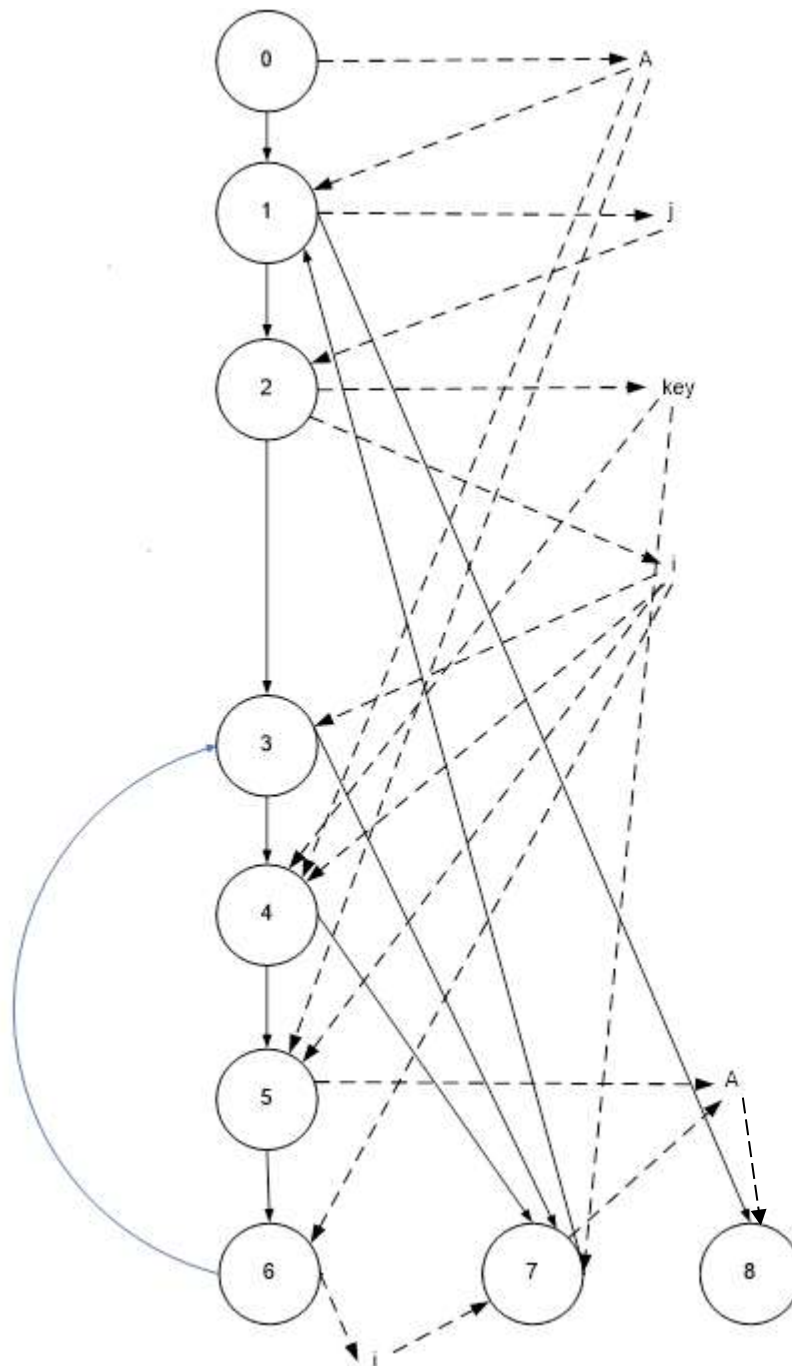


Рисунок 2.3 – Граф программы с управляющими и информационными связями

Шаг3а. Формирование множества определения данных:

DEF (0) ={A}

DEF (1) ={j}

DEF (2) ={key}

DEF (2) ={i}

DEF (5) ={A}

DEF (6) ={i}

DEF (7) ={A}

Шаг 3б. Формирование множества использования данных:

USE (1) ={A}

USE (2) ={j}

USE (3) ={i}

USE (4) = {A, key, i}

USE (5) ={A,i}

USE (6) ={i}

USE (7) = {i, key}

USE (8) ={A}

Шаг3в. Формирование цепочек определения и исполнения:

DU: {[A,0,1]; [A,0,4]; [A,0,5]; [A,5,8];

[j,1,2];

[key,2,4]; [key,2,7];

[i,2,3]; [i,2,4]; [i,2,5]; [i,2,6]; [i,6,7]; [i,6,3]; [i,6,4]; [i,6,5]}

Шаг 4. Формирование полного набора отрезков путей в управляющем графе (на основе построенных DU цепочек). Рисуется фрагмент информационного графа (ФИГ), а также соответствующий ему фрагмент управляющего графа (ФУГ). На рисунках 2.4÷2.9 представлено формирование полного набора отрезков путей в управляющем графе.

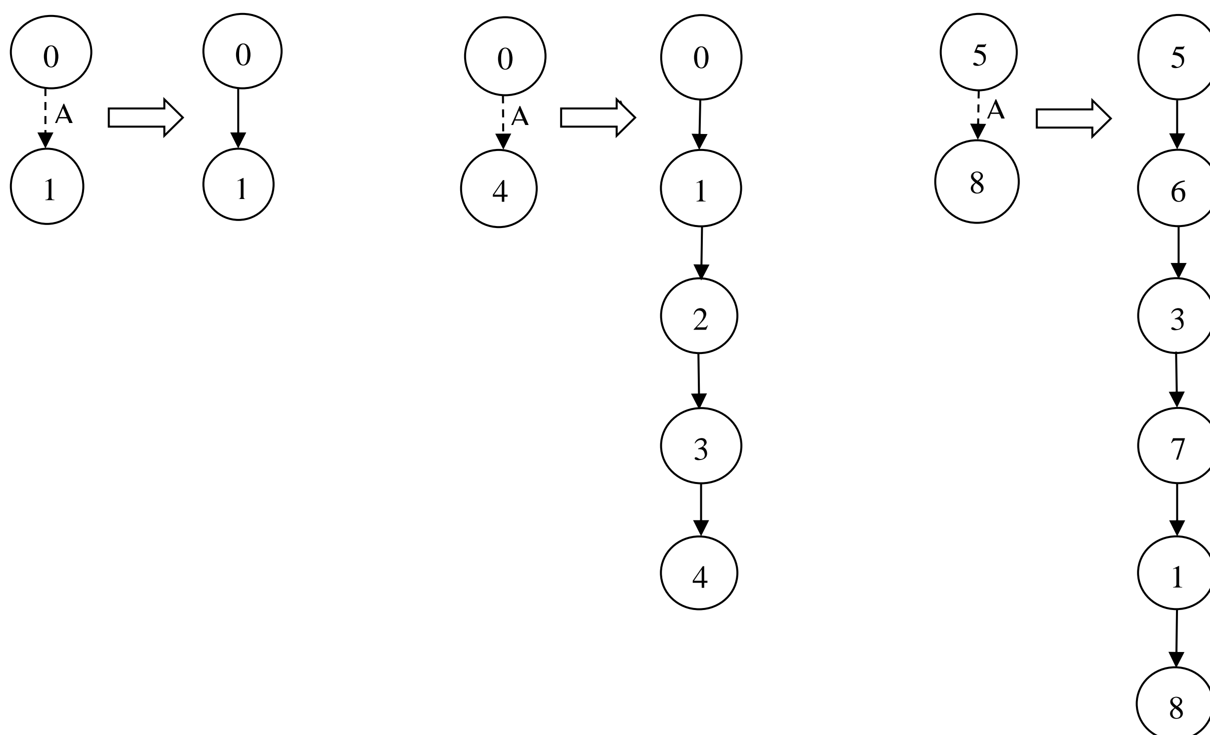


Рисунок 2.4 – [A, 0, 1] и [A, 0, 4], [A, 5, 8];

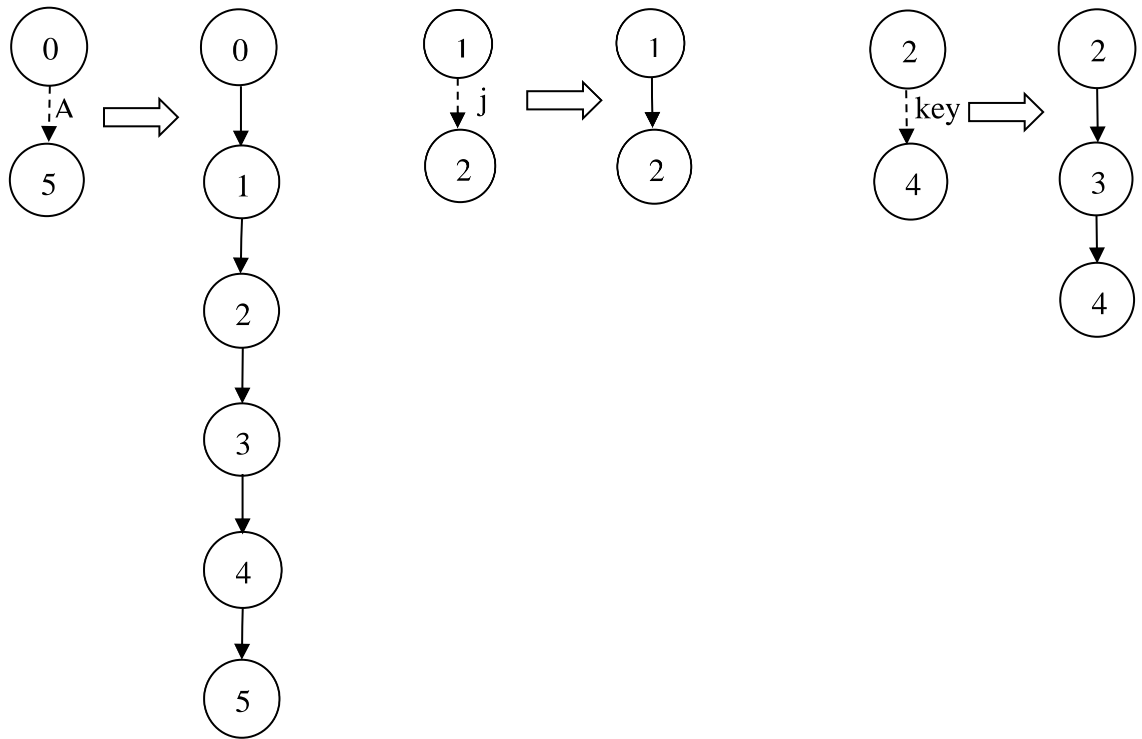


Рисунок 2.5 – [A, 0, 5], [j, 1, 2] и [key, 2, 4]

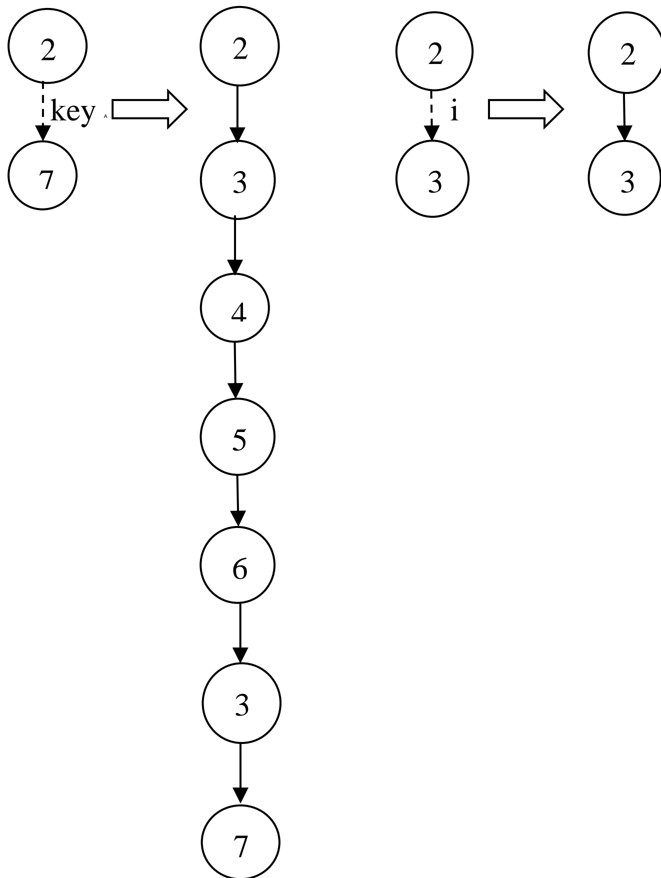
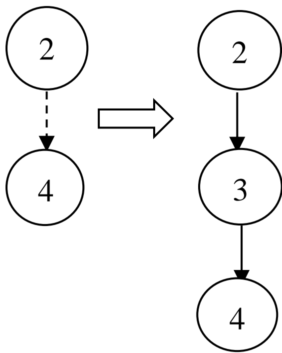


Рисунок 2.6 – [key, 2, 7] и [i, 2, 3]



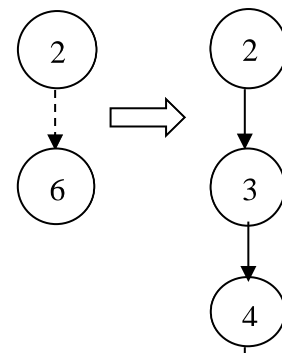
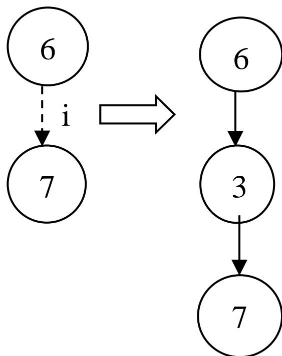


Рисунок 2.7 – [i, 2, 4] и [i, 2, 6]



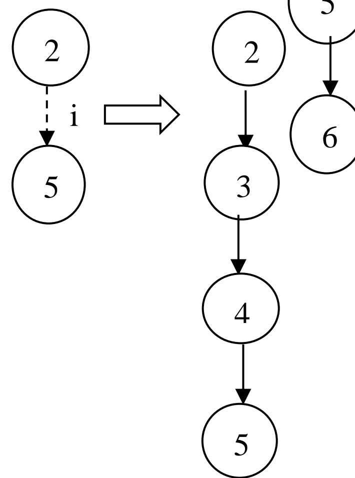
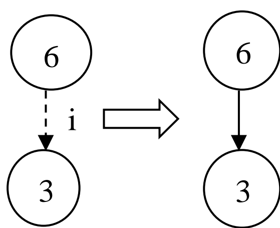
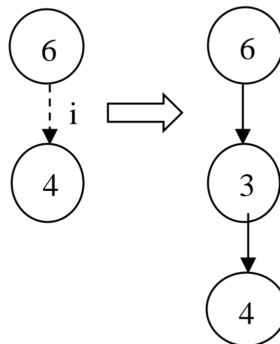


Рисунок 2.8 – [i, 6, 7] и [i, 2, 5]





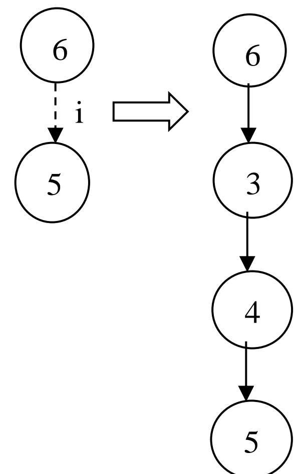


Рисунок 2.9 – [i, 6, 3], [i, 6, 4] и [i, 6, 4]

Шаг 5. Формирование маршрутов по цепочкам управляющего графа.

Путь 1: 0-1-2-3-4-5-6-3-7-1-8 // $n > 1$, A[i] > key массив не отсортирован.

Путь 2: 0-1-2-3-7-1-8 // $n > 1$, i=0, массив не отсортирован.

Путь 3: 0-1-2-3-4-7-1-8 // $n > 1$, A[i] < key массив отсортирован.

Путь 4: 0-1-2-3-4-5-6-3-4-5-6-3-7-1-8 // $n > 1$, режим нормальной обработки.

Шаг 6. В соответствии с маршрутами формируем тестовые варианты:

ТВ 1: Исходные данные: $n = 6$, $A = \{2, 5, 4, 6, 1, 3\}$ – вставка ключевого элемента на новое место в массиве, но не в начало массива.

Ожидаемый результат: $A = \{1, 2, 3, 4, 5, 6\}$.

ТВ 2: Исходные данные: $n = 6$, $A = \{2, 4, 5, 6, 1, 3\}$ – вставка ключевого элемента в начало массива.

Ожидаемый результат: $A = \{1, 2, 3, 4, 5, 6\}$.

ТВ 3: Исходные данные: $n = 6$, $A = \{1, 2, 3, 4, 5, 6\}$ – ключевой элемент остаётся на своём месте.

Ожидаемый результат: $A = \{1, 2, 3, 4, 5, 6\}$.

ТВ 4: Исходные данные: $n = 6$, $A = \{1, 2, 4, 5, 6, 3\}$ – повторный поиск места ключевого элемента в массиве.

Ожидаемый результат: $A = \{1, 2, 3, 4, 5, 6\}$.

Тестирование циклов

Типы циклов:

а) Простой цикл (рисунок 2.10).

Обычно тестируют следующими способами:

1) один раз пройти цикл.
2) два прохода через цикл.
3) проход всего цикла n раз.
4) m проходов цикла, где m<n
5) (n-1), n, (n+1) проходов цикла.

Пример подобного цикла на языке Ассемблер:

```
mov cx,n
cicle:
    adc [di], ax
    add si,2
    add di,2
    loop cicle
```

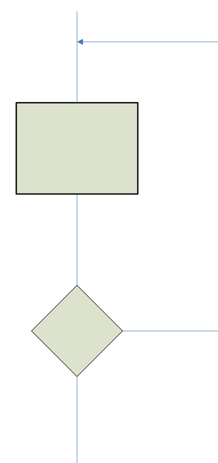


Рисунок 2.10 – Простой цикл

б) Вложенный цикл (рисунок 2.11).

Шаги тестирования:

Шаг 1. Выбирается самый внутренний цикл, при этом устанавливаются минимальные значения параметров для всех остальных циклов.

Шаг 2. Для внутреннего цикла проводятся тесты простого цикла, добавляются тесты исключенных значений и тесты значений, выходящих за пределы рабочего диапазона. Переходим к следующему по порядку объемлющему циклу. При этом сохраняются минимальные значения параметров для всех внешних объемлющих циклов и типовые значения параметров для всех вложенных циклов.

Шаг 3. Повторяем, пока не протестированы все циклы.

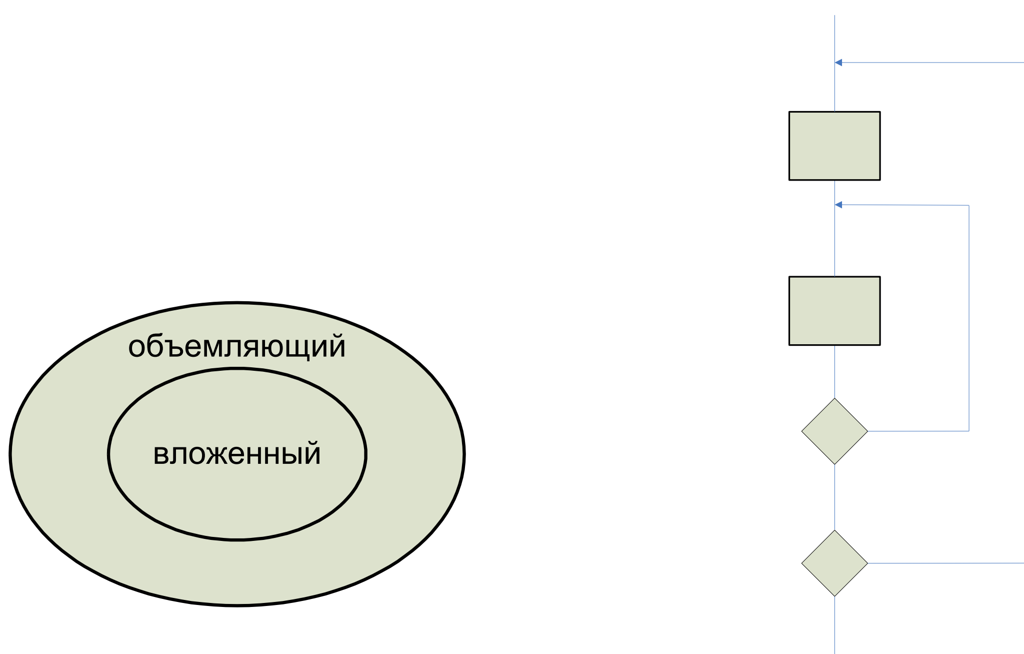


Рисунок 2.11 – Вложенный цикл

Пример подобного цикла на языке Ассемблер:

```
mov dx, Number_Of_Arrays
 loop_ext:
        call Print_Name_Of_Array
        mov cx,n
loop_in:    call Print_An_Array_Element
            dec cx
cmp cx,0
jz loop_in
dec dx
cmp dx,0
jnz loop_ext
```

в) Объединённые циклы (рисунок 2.12). Если циклы независимы друг от друга, тестируем отдельно два простых цикла. При наличии зависимости используется методика, аналогичная случаю вложенных циклов.

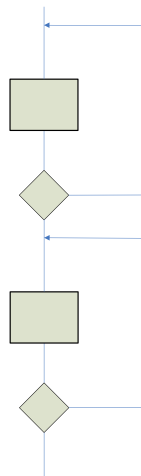


Рисунок 2.12 – Объединённые циклы

Пример двух независимых объединенных циклов на языке Ассемблер:

```
cicle: mov dl,slovo
        and dx,ax
       shl ax,1
       inc bx
             cmp dx,0
             jz fwd
             inc in_byte
             mov left_one,bl
             cmp in_byte,1
             jz right
             jnz fwd
             right: mov right_one,bl
 fwd:loop cicle
 mov ax,1
 mov cx,16
 mov bx,-1
 cicl:  mov dx,dslovo
             and dx,ax
       shl ax,1
       inc bx
             cmp dx,0
             jz fwd2
             inc in_byte2
             mov left_one2,bl
             cmp in_byte2,1
             jz right2
             jnz fwd2
             right2: mov right_one2,bl
 fwd2:loop cicl
```

г) Неструктурированные циклы (рисунок 2.13). Такие циклы переписываются в простые или вложенные. Такие циклы не подлежат тестированию.

Пример подобного цикла на языке Ассемблер:

```
      back:mov cx,n
   breaking_the_structure:cmp flag,0
   jz skip
   strange:cmp flag2,0
   jz back
skip: call Read_Line
push cx
cmp cx,n
jz strange
cmp flag3,0
jz breaking_the_structure
```

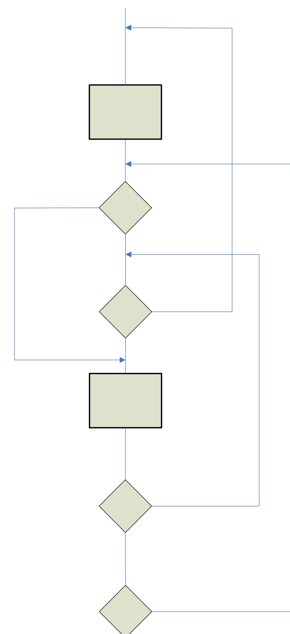


Рисунок 2.13 – Неструктурированные циклы

Такой стиль написания программ – признак непрофессионализма и отклонения от программистской этики.

Пример тестирования циклов

Рассмотрим тот же пример на базе сортировки вставкой. В программе сортировки вставкой встречается один вложенный цикл. Для начала тестируем внутренний цикл *while*. Для его тестирования на вход подаём неупорядоченный массив, содержащий больше трёх элементов. Затем тестируем объемлющий цикл, он также простой. Для его тестирования лучше запустить цикл на (n – 1), n, (n + 1) проход. То есть, при n = 10000, подготовить три тестовых варианта с 9999 элементами в первом, 10000 во втором и 10001 в третьем.

Контрольные вопросы

1. Чем характерно тестирования «белым ящиком»?
2. Какие существуют стратегии (способы) тестирования «белым ящиком»?
3. Что такое цикломатическая сложность и как она может быть вычислена?
4. Опишите алгоритм тестирования базового пути.
5. Перечислите и поясните типы условий, существующих в методе тестирования условий.
6. Назовите шаги алгоритма тестирования ветвей и операторов отношений.

7. Что представляют собой управляющий и информационный графы, и по какому принципу они строятся?
8. По какому принципу строятся DU цепочки?
9. Какие типы циклов вам известны?
10. Охарактеризуйте тестирования каждого из известных вам типов циклов.

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: Тестирование «чёрным ящиком». Функциональное тестирование

Цель работы: Тестирование программного продукта с помощью следующих способов тестирования «чёрным ящиком»:

a) Тестирование через разбиение на классы эквивалентности и анализ граничных условий.
b) Тестирование на основе диаграмм причин-следствий.

Содержание отчета:

1 Постановка задачи.
2 Описание предусловий и постусловий.
3 Дерево разбиения.
4 Тестовые варианты для первого способа тестирования «чёрным ящиком».
5 Перечисление причин и следствий.
6 Граф причинно-следственных связей.
7 Таблица решений.
8 Тестовые варианты для второго способа тестирования «чёрным ящиком».
9 Результаты работы программы.

Теоретическая часть

Одним из способов изучения поставленного вопроса является исследование стратегии тестирования, называемой стратегией «чёрного ящика», тестированием с управлением по данным, или тестированием с управлением по входу-выходу. При использовании этого способа программа рассматривается как «чёрный ящик». Иными словами, такое тестирование имеет целью выяснение обстоятельств, в которых поведение программы не соответствует ее спецификации. Тестовые же данные используются только в соответствии со спецификацией программы (т. е. без учета знаний о её внутренней структуре). При таком подходе обнаружение всех ошибок в программе является критерием исчерпывающего входного тестирования. Последнее может быть достигнуто, если в качестве тестовых наборов использовать все возможные наборы входных данных.

Для исчерпывающего тестирования определённых задач требуется бесконечное число тестов. Допустим, что делается попытка тестирования методом «чёрного ящика» интерпретатора с языка Java. Для построения исчерпывающего теста нужно использовать всё множество правильных программ на Java (фактически их число бесконечно) и все множество неправильных программ (т. е. действительно бесконечное число), чтобы убедиться в том, что компилятор обнаруживает все

ошибки. Только в этом случае синтаксически неверная программа не будет компилирована. Если же программа имеет собственную память (например, операционная система, база данных или система распределенных вычислений), то дело обстоит еще хуже. В таких программах исполнение команды (например, задание, запрос в базу данных, выполнение расчёта) зависит от того, какие события ей предшествовали, т. е. от предыдущих команд. Здесь следует перебрать не только все возможные команды, но и все их возможные последовательности. Тестирование «чёрного ящика» – это функциональное тестирование.

Из изложенного следует, что построение исчерпывающего входного теста невозможно. Это подтверждается двумя аргументами: во-первых, нельзя создать тест, гарантирующий отсутствие ошибок; во-вторых, разработка таких тестов противоречит экономическим требованиям. Поскольку исчерпывающее тестирование исключается, целью должна стать максимизация результативности капиталовложений в тестирование (иными словами, максимизация числа ошибок, обнаруживаемых одним тестом).

К способам тестирования «чёрного ящика» относятся следующие способы:

a) Эквивалентное тестирование (разбиение по классам эквивалентности).
b) Анализ граничных условий.
c) Тестирование на основе диаграмм причин-следствий.

Первые два способа будут объединены и рассмотрены совместно.

Эквивалентное тестирование и анализ граничных условий

Базовые понятия

Хороший тест имеет приемлемую вероятность обнаружения ошибки и, как уже отмечалось исчерпывающее входное тестирование программы невозможно. Следовательно, тестирование программы ограничивается использованием небольшого подмножества всех возможных входных данных. Тогда, конечно, хотелось бы выбрать для тестирования самое подходящее подмножество (т. е. подмножество с наивысшей вероятностью обнаружения большинства ошибок).

Правильно выбранный тест этого подмножества должен обладать двумя свойствами:

a) уменьшать, причем более чем на единицу, число других тестов, которые должны быть разработаны для достижения заранее определенной цели «приемлемого» тестирования;
b) покрывать значительную часть других возможных тестов, что в некоторой степени свидетельствует о наличии или отсутствии ошибок до и после применения, этого ограниченного множества значений входных данных.

Указанные свойства, несмотря на их кажущееся подобие, описывают два различных положения. Во-первых, каждый тест должен включать столько различных входных условий, сколько это возможно с тем, чтобы минимизировать общее число необходимых тестов. Во-вторых,

необходимо пытаться разбить входную область программы на конечное число классов эквивалентности так, чтобы можно было предположить (конечно, не абсолютно уверенно), что каждый тест, являющийся представителем некоторого класса, эквивалентен любому другому тесту этого класса. Иными словами, если один тест класса эквивалентности обнаруживает ошибку, то следует ожидать, что и все другие тесты этого класса эквивалентности будут обнаруживать ту же самую ошибку. Наоборот, если тест не обнаруживает ошибки, то следует ожидать, что ни один тест этого класса эквивалентности не будет обнаруживать ошибки (в том случае, когда некоторое подмножество класса эквивалентности не попадает в пределы любого другого класса эквивалентности, так как классы эквивалентности могут пересекаться).

Классы эквивалентности могут быть допустимыми – обозначают множество допустимых значений – и недопустимыми – обозначают множество недопустимых значений. Также классы эквивалентности можно подразделить на типы в соответствии с данными, которые они представляют:

a) определённое значение;
b) диапазон значений;
c) множество конкретных значений;
d) булево выражение (true, false).

При анализе граничных значений следует соблюдать следующие правила:

a) Если условия ввода заданы диапазоном значений $\{n..m\}$, то тестовые варианты должны быть построены для значений n и m, а также для значений немного меньших n (например, $n-1$) и немного больших m (например, $m+1$).
b) Если условия ввода заданы множество конкретных значений, то создаются тестовые варианты для проверки максимальных и минимальных из значений, и значений немного меньших минимальных и немного больше максимальных.
c) Если внутренняя структура данных имеет предписанные границы, то разрабатываются тестовые варианты, проверяющие эти границы.
d) Если входные и выходные данные являются упорядоченными множествами (файлы с последовательным доступом, таблицы, списки и т. п.), то следует тестировать обработку первого и последнего элемента в этих множествах.

Осуществление тестирования

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа:

a) выделение классов эквивалентности;
b) построение тестовых вариантов (тестов).

Для осуществления эквивалентного разбиения задача разбивается на предусловия и постусловия. Затем в рамках условий строится дерево разбиения, которое и поможет определить

классы эквивалентности. Второй шаг заключается в использовании классов эквивалентности для построения тестов, каждый лист дерева представляет собой отдельный тестовый вариант. Этот процесс включает в себя:

a) Назначение каждому классу эквивалентности уникального номера.
b) Проектирование новых тестов, каждый из которых покрывает как можно большее число непокрытых правильных классов эквивалентности, до тех пор, пока все правильные классы эквивалентности не будут покрыты тестами.
c) Запись тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы эквивалентности не будут покрыты тестами.

Данный способ тестирования представляется на примере всё того же алгоритма сортировки вставкой. Производится сортировка массива чисел по возрастанию значений (рисунок 3.1).

Итак, предусловия таковы:

— массив состоит из однородных элементов;
— массив ограничен размерностью *n*.

Эквивалентные разбиения:

— массив состоит из одного элемента;
— размерность массива от 2 до *n* элементов;
— размерность массива равна *n*.

Постусловия:

— массив упорядочен по возрастанию;
— массив не упорядочен.

Условимся, что $n = 1000$. Класс эквивалентности 3 покрывает класс эквивалентности 1 и 2, поэтому два тестовых варианта будут избыточными. Для наглядности перечислены все тестовые варианты:

— ТВ 1: Исходные данные: $n = 1$, $A = \{8\}$;
— Ожидаемый результат: $A = \{8\}$; массив упорядочен.
— ТВ 2: Исходные данные: $n = 1000$, $A = \{5, 12, 8, 256, \ldots, 90\}$;
— Ожидаемый результат: $A = \{\ldots\}$; массив упорядочен.
— ТВ 3: Исходные данные: $n = 6$, $A = \{1, 5, 6, 3, 256, 9\}$;
— Ожидаемый результат: $A = \{1, 3, 5, 6, 9, 256\}$; массив упорядочен.
— ТВ 4: Исходные данные: $n = 7$, $A = \{1, 2, 6, 4, \%, 20, 11\}$;
— Ожидаемый результат: *Error 1* – встречен нечисловой символ; массив не упорядочен.
— ТВ 5: Исходные данные: $n = 0$, $A = \{\}$;

— Ожидаемый результат: *Error 2* – длина массива $A = 0$, сортировка невозможна; массив не упорядочен.

— ТВ 6: Исходные данные: $n = 4$, $A = \{\}$;

— Ожидаемый результат: *Error 3* – массив A пуст, сортировка невозможна; массив не упорядочен.

— ТВ 7: Исходные данные: $n = 3$, $A = \{13, 7, 8, 7, 9, 10, 22, 58\}$;

— Ожидаемый результат: *Error 4* – длина массива $A > n$; массив не упорядочен.

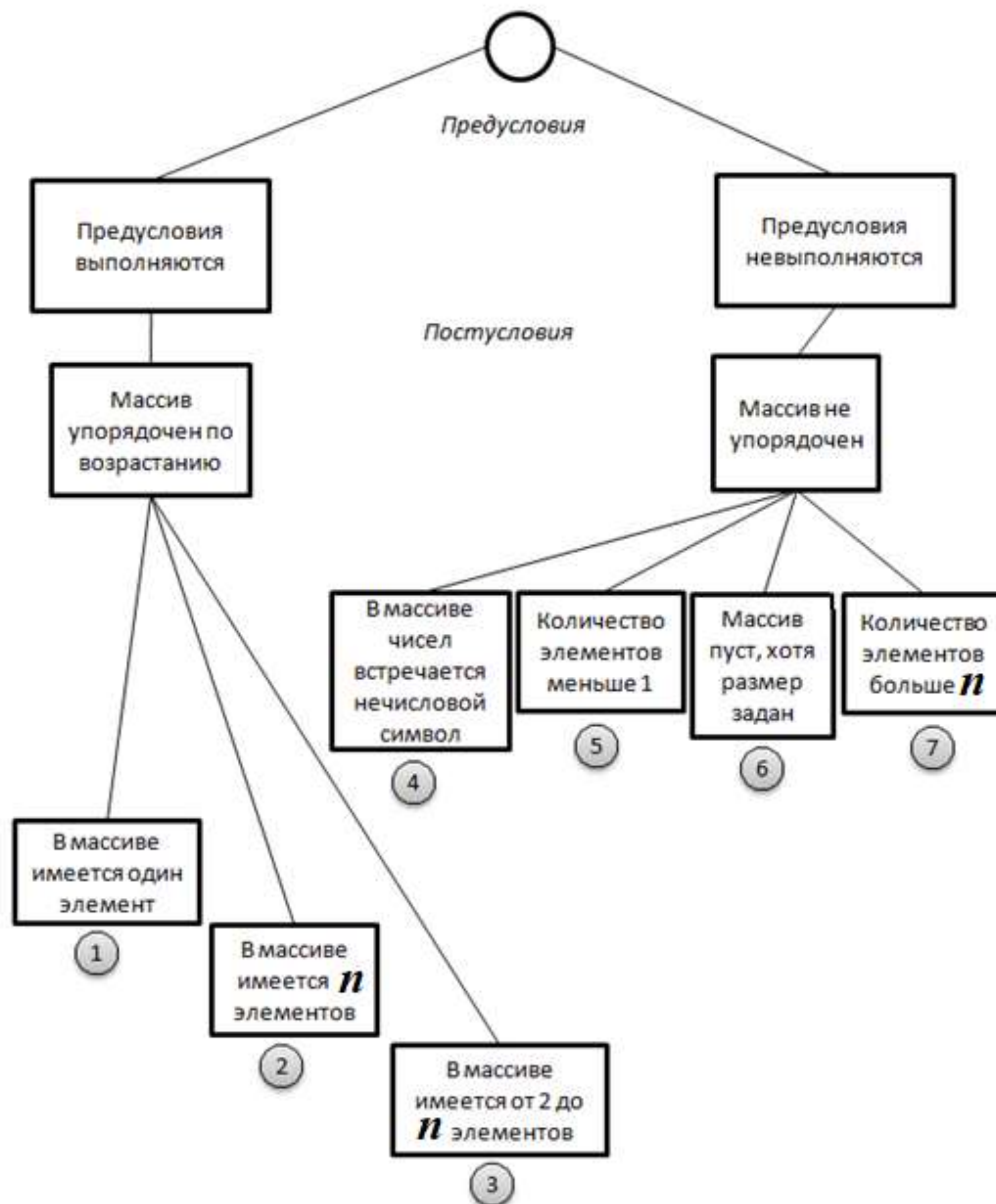


Рисунок 3.1 – Дерево разбиения для процедуры Insertion_Sort

Тестирование при помощи диаграмм причин–следствий

Базовые понятия

Диаграммы причинно-следственных связей — способ проектирования тестовых вариантов, который обеспечивает формальную запись логических условий и соответствующих действий. Используется автоматный подход к решению задачи.

Шаги тестирования

Шаг 1. Для каждого модуля перечисляются причины (условия ввода или классы эквивалентности условий ввода) и следствия (действия или условия вывода). Каждой причине и следствию присваивается свой идентификатор.

Шаг 2. Разрабатывается граф причинно-следственных связей.

Шаг 3. Граф преобразуется в таблицу решений.

Шаг 4. Столбцы таблицы решений преобразуются в тестовые варианты.

Изобразим базовые символы для записи графов причин и следствий (cause-effect graphs). Сделаем предварительные *замечания:*

- причины будем обозначать символами c_j, а следствия — символами e_i;
- каждый узел графа может находиться в состоянии 0 или 1 (0 — состояние отсутствует, 1 — состояние присутствует).

Функция *тождество* (рисунок 3.2) устанавливает, что если значение c_1 есть 1, то и значение e_1 есть 1; в противном случае значение e_1 есть 0.

Рисунок 3.2 – Функция *тождество*

Функция *не* (рисунок 3.3) устанавливает, что если значение c_1 есть 1, то значение e_1 есть 0; в противном случае значение e_1 есть 1.

Рисунок 3.3 – Функция *не*

Функция *или* (рисунок 3.4) устанавливает, что если значение c_1 или c_2 есть 1, то e_1 есть1, в противном случае e_1 есть 0.

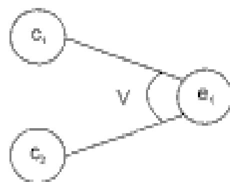


Рисунок 3.4 – Функция *или*

Функция *и* (рисунок 3.5) устанавливает, что если и c_1 и c_2 есть 1, то e_l есть 1, в противном случае e_1 есть 0.

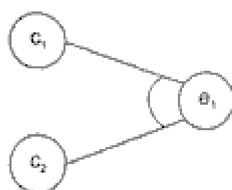


Рисунок 3.5 – Функция *и*

Часто определенные комбинации причин невозможны из-за синтаксических или внешних ограничений. Используются перечисленные ниже обозначения ограничений.

Ограничение E (исключает, Exclusive, рисунок 3.6) устанавливает, что E должно быть истинным, если хотя бы одна из причин — *a* или *b* — принимает значение 1 (*a* и *b* не могут принимать значение 1 одновременно).

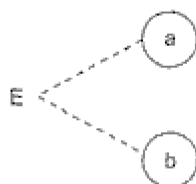


Рисунок 3.6 – Ограничение E (исключает, Exclusive)

Ограничение I (включает, Inclusive, рисунок 3.7) устанавливает, что по крайней мере одна из величин, *a, b,* или *c,* всегда должна быть равной 1 *(a, b и c* не могут принимать значение 0 одновременно).

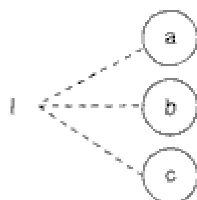


Рисунок 3.7 – Ограничение I (включает, Inclusive)

Ограничение O (одно и только одно, Only one, рисунок 3.8) устанавливает, что одна и только одна из величин, a или b, должна быть равна 1.

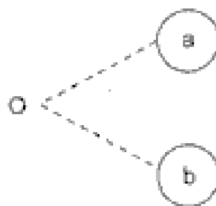


Рисунок 3.8 – Ограничение O (одно и только одно, Only one)

Ограничение R (требует, Requires, рисунок 3.9) устанавливает, что если a принимает значение 1, то и b должна принимать значение 1 (нельзя, чтобы a было равно 1, а b — 0).

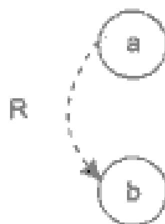


Рисунок 3.9 – Ограничение R (требует, Requires)

Часто возникает необходимость в **ограничениях для следствий**.

Ограничение M (скрывает, Masks, рисунок 3.10) устанавливает, что если следствие a имеет значение 1, то следствие b должно принять значение 0.

Рисунок 3.10 – Ограничение M (скрывает, Masks)

Для иллюстрации использования способа рассмотрим пример, когда программа выполняет расчет оплаты электричества по среднему или переменному тарифу.

При расчете по среднему тарифу:

- при месячном потреблении энергии меньшем, чем 100 кВт/ч, выставляется фиксированная сумма;
- при потреблении энергии большем или равном 100 кВт/ч применяется процедура *A* планирования расчета.

При расчете по переменному тарифу:

— при месячном потреблении энергии меньшем, чем 100 кВт/ч, применяется процедура *A* планирования расчета;
— при потреблении энергии большем или равном 100 кВт/ч применяется процедура *B* планирования расчета.

Шаг 1. Причинами являются:

расчет по среднему тарифу (1);

расчет по переменному тарифу (2);

месячное потребление электроэнергии меньшее, чем 100 кВт/ч (3);

месячное потребление электроэнергии большее или равное 100 кВт/ч (4).

На основе различных комбинаций причин можно перечислить следующие следствия:

минимальная месячная стоимость (101);

процедура *A* планирования расчета (102);

процедура *B* планирования расчета (103).

Щаг 2. Разработка графа причинно-следственных связей (рисунок 3.11).

Узлы причин перечислим по вертикали у левого края рисунка, а узлы следствий — у правого края рисунка. Для следствия 102 возникает необходимость введения вторичных причин — 11 и 12, — их размещаем в центральной части рисунка.

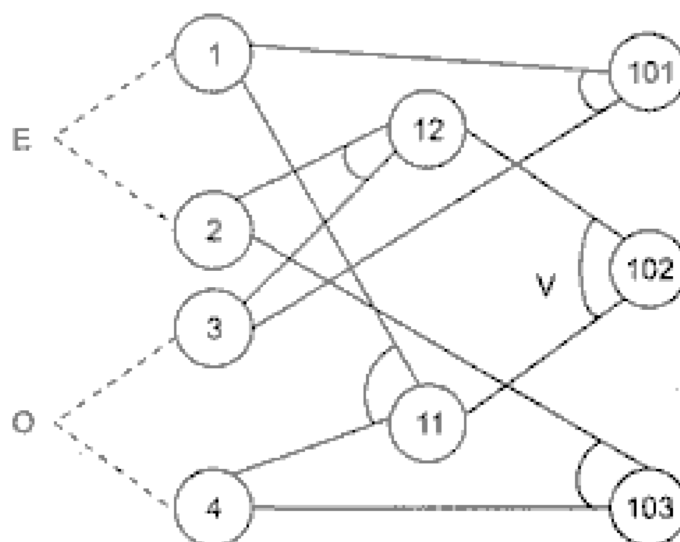


Рисунок 3.11 – Граф причинно-следственных связей

Шаг 3. Генерация таблицы решений. При генерации причины рассматриваются как условия, а следствия — как действия.

Порядок генерации.

Шаг 3.1. Выбирается некоторое следствие, которое должно быть в состоянии «1».

Шаг 3.2. Находятся все комбинации причин (с учетом ограничений), которые устанавливают это следствие в состояние «1». Для этого из следствия прокладывается обратная трасса через граф.

Шаг 3.3. Для каждой комбинации причин, приводящих следствие в состояние «1», строится один столбец.

Шаг 3.4. Для каждой комбинации причин доопределяются состояния всех других следствий. Они помещаются в тот же столбец таблицы решений.

Шаг 3.5. Действия 3.1 ÷ 3.4 повторяются для всех следствий графа.

Таблица решений для этого примера показана в таблица 3.1.

Таблица 3.1 – Таблица решений для расчёта оплаты электричества

Номера столбцов →			1	2	3	4
Условия	**Причины**	1	1	0	1	0
		2	0	1	0	1
		3	1	1	0	0
		4	0	0	1	1
	Вторичные причины	11	0	0	1	0
		12	0	1	0	0
Действия	**Следствия**	101	1	0	0	0
		102	0	1	1	0
		103	0	0	0	1

Шаг 4. Преобразование каждого столбца таблицы в тестовый вариант. В нашем примере таких вариантов четыре:

ТВ 1: Исходные данные: расчёт по среднему тарифу; месячное потребление электроэнергии 75 кВт/ч.

Ожидаемый результат: минимальная месячная стоимость.

ТВ 2: Исходные данные: расчёт по переменному тарифу; месячное потребление электроэнергии 90 кВт/ч.

Ожидаемый результат: процедура *А* планирования расчёта.

ТВ 3: Исходные данные: расчёт по среднему тарифу; месячное потребление электроэнергии 100 кВт/ч.

Ожидаемый результат: процедура *А* планирования расчёта.

ТВ 4: Исходные данные: расчет по переменному тарифу; месячное потребление электроэнергии 100 кВт/ч.

Ожидаемый результат: процедура *B* планирования расчета.

Пример тестирования с помощью диаграмм причин-следствий

Шаг 1. Причинами являются:

1) введён массив не чисел.
2) верхняя граница размера массива >10000 ($n>10000$).
3) строго возрастающий массив чисел.
4) строго убывающий массив чисел.
5) неупорядоченный (произвольный) массив чисел.
6) массив из одного и того же числа при $n \geq 2$ & $n \leq 10000$

Можно перечислить следующие следствия:

11) сообщение об ошибке, массив не сортируется (ERROR);
12) печать отсортированного массива.

Шаг 2. Разработка графа причинно-следственных связей (рисунок 3.12).

Выбранный тип ограничения для всех причин – Only one (***O***). Следствия образуются через функции *или* (V).

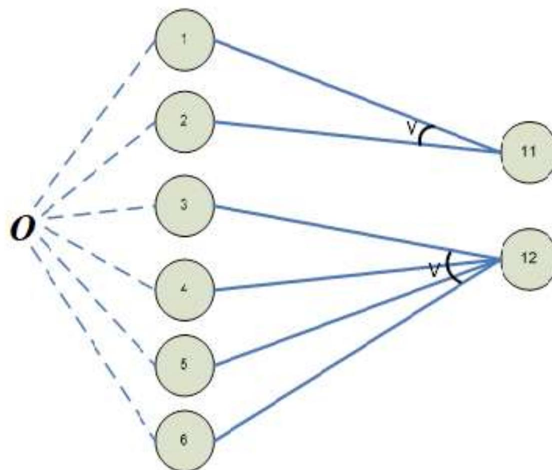


Рисунок 3.12 – Граф причинно-следственных связей

Шаг 3. Генерация таблицы решений (таблица 3.2).

Таблица 3.2 – Таблица решений

Номера столбцов →			1	2	3	4	5	6
Условия	**Причины**	1	1	0	0	0	0	0
		2	0	1	0	0	0	0
		3	0	0	1	0	0	0
		4	0	0	0	1	0	0
		5	0	0	0	0	1	0
		6	0	0	0	0	0	1
Действия	**Следствия**	11	1	1	0	0	0	0
		12	0	0	1	1	1	1

Шаг 4. Преобразование каждого столбца таблицы 3.2 в тестовый вариант (таблица 3.3):

Таблица 3.3 – Таблица тестовых вариантов

№ ТВ	**Исходные данные**	**Ожидаемый результат**
1	$n = 5$, $A = \{a, b, c, 1, d\}$	ERROR
2	$n = 10001$, $A = \{\ldots\}$ – большой массив	ERROR
3	$n = 6$, $A = \{1, 2, 5, 9, 12, 13\}$	$A = \{1, 2, 5, 9, 12, 13\}$
4	$n = 5$, $A = \{5, 4, 3, 2, 1\}$	$A = \{1, 2, 3, 4, 5\}$
5	$n = 7$, $A = \{1, 9, 2, 1, 2, 3, 4, 6\}$	$A = \{1, 1, 2, 2, 3, 4, 6, 9\}$
6	$n = 7$, $A = \{9, 9, 9, 9, 9, 9, 9\}$	$A = \{9, 9, 9, 9, 9, 9, 9\}$

Контрольные вопросы

1. Чем характерно тестирования «чёрным ящиком»?
2. Какие существуют стратегии (способы) тестирования «чёрным ящиком»?
3. Какие существуют виды классов эквивалентности?
4. Какова суть тестирования через эквивалентное разбиение и анализ граничных условий?
5. Что такое диаграммы причинно-следственных связей, и из каких элементов они состоят?
6. Как происходит построение графа причинно-следственных связей?
7. Как составляется таблица решений для способа тестирования через диаграммы причин-следствий?

ЛАБОРАТОРНАЯ РАБОТА № 4 (1)

Тема: Автоматизация процесса тестирования программного продукта.

Цель работы: Разработка драйвера тестирования в соответствии с лабораторными работами № 1 и № 2.

Содержание отчёта:

a) Алгоритм работы драйвера тестирования.

b) Варианты тестирования (исходные данные, ожидаемые результаты).

c) Результат работы драйвера:

   1) вывод номера варианта тестирования;
   2) исходные данные;
   3) ожидаемые результаты;
   4) реальные результаты;
   5) маршруты тестирования.

d) Приложение. Листинг программы.

Теоретическая часть

Драйвер (модуль для тестирования) – управляющая программа, которая принимает исходные данные (InData) и ожидаемые результаты (ExpRes) тестовых вариантов. Она запускает в работу тестируемый модуль, получает из модуля реальные результаты (OutData) и формирует отчёт о тестировании. Таким образом драйверы тестирования используются для автоматизации процесса тестирования.

Основной объем тестирования практически любой сложной системы обычно выполняется в автоматическом режиме. Кроме того, тестируемая система обычно разбивается на отдельные модули, каждый из которых тестируется вначале отдельно от других, а затем в комплексе. Это означает, что для выполнения тестирования необходимо создать некоторую среду, которая обеспечит запуск и выполнение тестируемого модуля, передаст ему входные данные, соберёт реальные выходные данные, полученные в результате работы системы на заданных входных данных. После этого среда должна сравнить реальные выходные данные с ожидаемыми и на основании данного сравнения сделать вывод о соответствии поведения модуля заданному поведению (рисунок 4.1).

Рисунок 4.1 – Драйвер в обобщённой среде тестирования

Тестовое окружение для программного кода на структурных языках программирования состоит из двух компонентов – драйвера и заглушек. Драйвер обеспечивает запуск и выполнение тестируемого модуля и заглушек, которые моделируют функции, вызываемые из данного модуля. Разработка тестового драйвера представляет собой отдельную задачу тестирования, сам драйвер должен быть протестирован, дабы исключить неверное тестирование. Драйвер и заглушки могут иметь различные уровни сложности, требуемый уровень сложности выбирается в зависимости от сложности тестируемого модуля и уровня тестирования. Так, драйвер может выполнять следующие функции:

a) вызов тестируемого модуля;
b) передача в тестируемый модуль входных значений и прием результатов;
c) вывод выходных значений;
d) протоколирование процесса тестирования и ключевых точек программы.

Функции заглушек:

a) не производить никаких действий (такие заглушки нужны для корректной сборки тестируемого модуля);
b) вызова;
c) вывод сообщений со значениями Вывод сообщений о том, что заглушка была параметров, переданных в функцию;
d) возвращение значения, заранее заданного во входных параметрах теста;
e) вывод значения, заранее заданного во входных параметрах теста;
f) приём от тестируемого программного обеспечения значений и передача их в драйвер.

Для тестирования программного кода, написанного на процедурном языке программирования, используются драйверы, представляющие собой программу с точкой входа (к примеру, функцией *main*()), функциями запуска тестируемого модуля и функциями сбора результатов. Обычно драйвер имеет как минимум одну функцию – точку входа, которой передаётся управление при его вызове.

Функции-заглушки могут помещаться в тот же файл исходного кода, что и основной текст драйвера. Имена и параметры заглушек должны совпадать с именами и параметрами “заглушаемых” функций реальной системы. Это требование важно чтобы максимально точно моделировать поведение реальной системы по передаче данных.

Рассмотри подробно этапы создания драйвера тестирования.

Этап 1. Генерация входных данных

Предположим, что тестируется некоторый компонент, который в качестве входных данных принимает n полей. Для тестирования этого компонента идеально было бы сгенерировать всевозможные входные данные. Примерно это выглядит так:

Компоненту можно послать валидное значение поля, невалидное значение и пустое поле. Для позитивного тестирования входные данные состояли бы из 2^n строк (генерация всех возможных комбинаций с валидным и пустым значением поля). Однако на практике тестирование компонента заняло бы очень много времени, поэтому часто необходимо сокращать входные данные. Для этого обычно генерируются не всевозможные варианты входных данных, а только часть из них. Если наш компонент принимает m обязательных полей и k опциональных, то может сгенерировать 2^m комбинаций и для каждой из них подставить какие-то значения из k опциональных полей.

Для компонента, принимающего 4 полей входные данные, будут выглядеть так:

```
FIELD1 FIELD2 FIELD3
---           ---           ---
Valid_data    Valid_data    Valid_data
Valid_data    Valid_data    ---
Valid_data    ---           Valid_data
---           Valid_data    Valid_data
Valid_data    ---           ---
---           ---           Valid_data
Valid_data    Valid_data    Valid_data
```

Для негативного тестирования, однако, входные данные могут расшириться в виду того, что должны послать не валидные данные разных типов (например, String, Integer, Decimal и т. д.)

Этап 2. Алгоритм драйвера

Алгоритм драйвера предельно прост. Работа драйвера заключается в последовательном считывании входных данных, отправке их тестируемому компоненту, получении результата и сравнении его с ожидаемым результатом. При этом если действительный результат не соответствует ожидаемому, то статус драйвера FAIL. В драйвере также необходимо логировать все результаты, результаты каждой итерации записывать в выходной файл и ее статус. Если результат хоть одной итерации FAIL, то драйвер должен продолжать свою работу. Таким образом в конце можно проанализировать все данные и выявить потенциальные дефекты. Сказанное выглядит примерно так:

```
Bool script_status = true
Matrix M = GenerateInputData()
For i ← 1 to M.rows do
        Result expected = GetExptected(M[i])
        Result actual = SendRequest(M[i])
        Bool status = Compare(expected, actual)
        If status = false
        Then sctipt_status = false
        LogResult(file,i,actual, expected,status)
LogScriptStatus(script_status)
```

Функция GenerateInputData() генерирует входные данные. Далее каждая строка из водных данных считывается и посылается тестируемому компоненту. Предварительно генерируем ожидаемый результат для данных входных данных и после сравниваем реальный результат с ожидаемым. В лог-файл на каждой итерации записываем номер итерации, реальный и ожидаемый результат и статус. Если на какой-то итерации результаты не совпадают, то драйвер будет иметь статус FAIL. Для этого необходимо определить логическую переменную script_status, которую запишем в конец лог-файла.

В качестве примера драйвера и заглушек можно рассмотреть реализацию стека на языке C, причём значения, помещаемые в стек, хранятся не в оперативной памяти, а помещаются в ПЗУ при помощи отдельного модуля, содержащего две функции – записи данных и чтения данных. Формат этих функций таков:

```
void NV_Read(char *destination, long length, long offset);
void NV_Write(char *source, long length, long offset);
```

Здесь *destination* – адрес области памяти, в которую записывается считанное значение, *source* – адрес области памяти, из которой читается значение, *length* – длина записываемой области памяти, *offset* – смещение относительно начального адреса памяти.

Реализация стека с использованием этих функций выглядит следующим образом:

```
long currentOffset;
void initStack()
{
	currentOffset=0;
}
void push(int value)
{
	NV_Write((int*)&value,sizeof(int),currentOffset);
	currentOffset+=sizeof(int);
}
int pop()
{
	int value;
	if (currentOffset>0)
	{
		NV_Read((int*)&value,sizeof(int),currentOffset;
		currentOffset-=sizeof(int);
	}
}
```

При выполнении этого кода на реальной системе происходит запись в ПЗУ, однако, если хотим протестировать только реализацию стека, изолировав её от реализации модуля работы с памятью, необходимо использовать заглушки вместо реальных функций.

Для имитации работы ПЗУ можно выделить достаточно большой участок оперативной памяти, в которой и будет производиться запись данных, получаемых заглушкой. Заглушки для функций могут выглядеть следующим образом:

```
char nvrom[1024];

void NV_Read(char *destination, long length, long offset)
{
	printf("NV_Read called\n");
	memcpy(destination, nvrom+offset, length);
}
void NV_Write(char *source, long length, long offset);
{
	printf("NV_Write called\n");
	memcpy(nvrom+offset, source, length);
}
```

Каждая из заглушек выводит трассировочное сообщение и перемещает переданное значение в память, эмулирующую ПЗУ (функция NV_Write) или возвращает по ссылке значение, хранящееся в памяти, эмулирующей ПЗУ (функция NV_Read).

Схема взаимодействия тестируемого ПО (функций работы со стеком) с тестовым окружением (драйвером и заглушками функций работы с ПЗУ) изображена на рисунке 4.2.

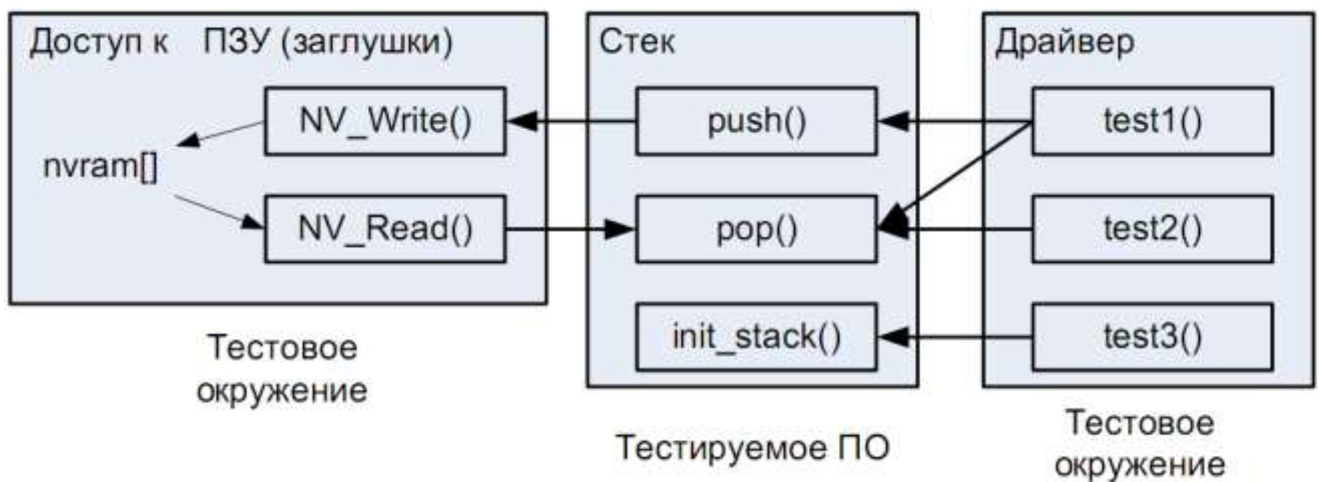


Рисунок 4.2 – Схема взаимодействия тестового окружения и тестируемого ПО

Пример тестирования веб-страницы

В последнее время разрабатывается огромное количество веб-страниц и появляется необходимость протестировать работоспособность всех компонентов с данной страницы. Естественно, веб-страницы постоянно расширяются и поэтому необходимо постоянно обновлять и расширять свой драйвер для тестирования. Изменять и дополнять сам код драйвера работа трудоемкая. Поэтому используется следующий подход: создается библиотека функции для тестирования веб-страницы: переход по ссылке, проверка загрузки страницы, клик по компоненту, ввод поля, логин, логаут и т. д. Далее создается входная матрица, в которую записывается все действия, которые предпринимаются для тестирования, эти действия основаны на описанных нами библиотечных функциях. Таким образом при обновлении страницы нет необходимости обновлять сам драйвер, просто дополняем данную матрицу и репозитории объектов. Пример входной матрицы:

Include	Action	URL	Name	Pwd	Link	Text
Y	Navigate	www.web.com				
Y	Login		User	Password		
Y	Click				My Page	
Y	Click				Create message	
Y	Edit				InputBox	Hello World
N	Edit				InputBox	This will not be executed
N	Edit				InputBox	This will not be executed
Y	Click				Send message	
N	Logaut					
Y	Click				My page	
Y	Logaut					

Колонка Include содержит Y/N, при желании можем временно пропустить выполнение определенного шага. Action – это название вызываемой функции. Name и Pwd – это поля для

определенной нами функции Login. Link содержит ссылки, по которым будет щелкать драйвер. Text – это текст который будет вводится в определенный Inputbox.

Репозитории объектов – это некоторый модуль, в котором описаны все классы, с которыми работаем (My_page, Send message, Create message и т. д.). Далее в данном драйвере описываем обобщенные функции, которые выполняют определенное действие для объектов всех классов. (Специальные средства автоматического тестирования типа Quick Test Professional позволяют автоматически считывать с любого окна объекты в репозитории).

Драйвер будет содержать функцию Matrix_Parser, которая считывает последовательно строки из матрицы и вызывает соответствующие функции для определенных объектов.

```
M ← InputMatrix()
For i ← 1 to M.Length
       MatrixParsing(i)

MatrixParsing(int i)
       if M[i][1] == 'N'
              then return
       else
              void (*fptr)(void)  // Определяем указатель на функцию (или делегат C#)
              fptr = FabricFunction(M[i][2])
              switch(M[i][2])
              "Navigate":
                     Begin
                            fptr(M[i][3])
                            LogAction(i, M[i][2])
                     end
               "Click":
                     Begin
                            fptr(M[i][6])
                            LogAction(i, M[i][2])
                     end
               "Edit":
                     Begin
                            fptr(M[i][7])
                            LogAction(i, M[i][2])
                     End
               "Login":
                     Begin
                            fptr(M[i][4], M[i][5])
                            LogAction(i, M[i][2])
                     end
              "Logout":
                     Begin
                            fptr()
                            LogAction(i, M[i][2])
                     End
```

FabricFunction(char *) – это фабричная функция, которая в зависимости от входного параметра возвращает указатель на определенную функцию.

LogAction() – функция логирования, которая записывает в лог-файл все наши действия.

Этапы работы

a) Разработка драйвера тестирования в соответствии со способами тестирования «белого ящика».
b) Разработка драйвера тестирования в соответствии со способами тестирования «черного ящика».
c) Объединение драйверов в единую тестирующую программу.
d) Анализ результатов.

Контрольные вопросы

1. Что такое драйвер?
2. Каковы функции драйвера?
3. Что такое тестовое окружение?
4. Что такое заглушки и какие функции они выполняют?
5. По какому принципу происходит построение драйвера для тестирования «белым ящиком» и «чёрным ящиком»?

ЛАБОРАТОРНАЯ РАБОТА № 4 (2)

Тема: Создание unit тестов

Цель работы: Разработка unit тестов для задания, представленного в лабораторной работе №1.

Содержание отчёта:

a) Алгоритм unit теста.

b) Варианты тестирования (исходные данные, ожидаемые результаты).

c) Результат работы (скриншоты):

1) вывод номера варианта тестирования;
2) исходные данные;
3) ожидаемые результаты;
4) реальные результаты.

d) Приложение. Листинг программы.

Введение

NUnit это утилита для тестирования программ, написанных на языках DOT .NET. Данная утилита предоставляет nunit-framework для написания юнит – тестов (unit tests) и визуальную среду для прогона юнит - тестов. Данная утилита была создана на основе раннее написанной аналогичной программы JUnit, которая предоставляла те же функции для Java платформы. Сама утилита была написана на C#. Данная утилита распространяется открытым кодом (Open Source). Скачать утилиту можно на официальном сайте.

Официальный сайт: [http://www.nunit.org]

Ссылка для скачивания [http://www.nunit.org/index.php?p=download]

В данной лабораторной работе будут описаны шаги создания юнит-тестов их прогон на примере простой программы.

Используемое программное обеспечение:

Visual C# Express edition.

NUnit 2.4.8

Подойдёт любая версия Visual Studio, вышедшая после 2003 года.

Создание юнит-тестов

Создание юнит тестов будет состоять из следующих шагов: в начале пишется класс, который будет тестироваться. Затем класс компилируется в .dll файл. Далее создаётся непосредственно юнит – тест. В данный юнит тест добавляется ссылка на .dll файл и ссылка на nunit-framework.

Шаг 1. Создание тестируемого класса:

В Visual Studio создаём новый проект на языке C# (рисунок 4.1)

File -> New Project

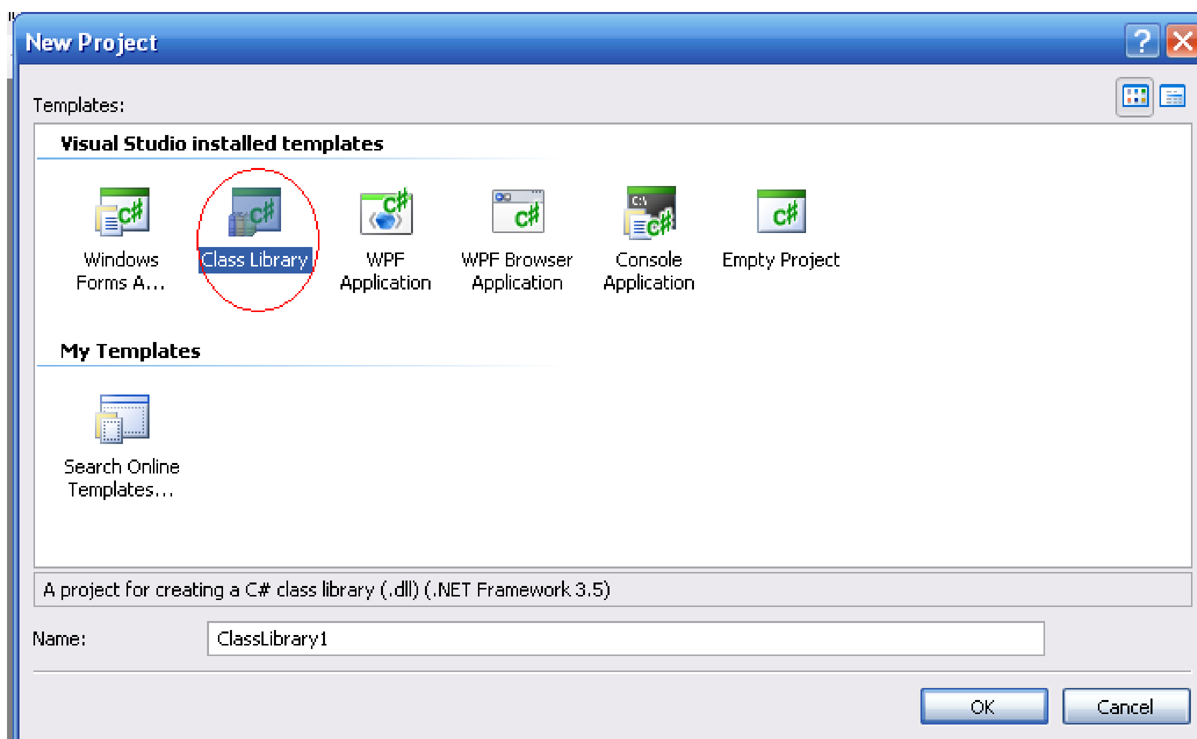


Рисунок 4.1 - Выбор типа проекта

Теперь можно непосредственно приступить к написанию алгоритма. Алгоритм будет заключаться в том, что будет переставлять непересекающиеся части массива одинакового размера. Основной метод – RepartArray. В него будут передаваться массив, начало первого массива, начало второго массива и их размер. И возвращаться переставленный массив. В случае пересекающихся частей массивов будем вызывать ошибку

Алгоритм:

```
1 if Index2+Length>masLength
2       goto retunr
3 if Index1+Length>Index2
4       goto retunr
5 while i>Length
6       a[Index1+i]<->a[Index2+i]
```

```
6       i->i+1
7 endloop
8 retunr
```

Реализация алгоритма на C#:

```
    /// <summary>
        /// tested class
        /// </summary>
        public class Clas1
        {
            /// <summary>
            /// default Constructor
            /// </summary>
            public Class1() { }
            /// <summary>
            /// Best Method
            /// </summary>
            /// <param name="mas"></param>
            /// <param name="Index1"></param>
            /// <param name="Index2"></param>
            /// <param name="Length"></param>
            /// <returns></returns>
            public int[] RepartArray(int[] mas,int Index1,int Index2,int Length)
            {
                int[] a = (int[])mas.Clone();
                if (Index2 + Length > a.Length) throw new ArgumentException("Index2 is too
large");
                if (Index1 + Length > Index2) throw new ArgumentException("Arrays are
crossed");
                int i = 0;
                int temp = 0;
                while (i < Length)
                {
                    temp = a[Index1 + i];
                    a[Index1 + i] = a[Index2 + i];
                    a[Index2+i]=temp;
                    i++;
                }
                return a;
            }
        }
```

Комментарии не обязательны, но лучше их оставить. Они сильно облегчают работу.

Теперь должны откомпилировать проект в .dll файл, это можно сделать, нажав в меню Build->Build Solution. Если нет ошибок, то внизу на панели должна быть надпись Build succesed. На этом создание класса закончено.

Шаг 2. Создание юнит-тестов

Юнит тесты — это программы, которые тестируют другие программы. Классы, которые тестируют программы, должны быть помечены атрибутом `[TestFixture.` Методы, которые тестируют программу, должны быть помечены атрибутом `[Test]`.

Для сравнения результатов программы с ожидаемыми результатами существует специальный статический класс Assert из пространства имён NUnit.Framework. Методы данного класса будут показателем того, пройден тест или нет.

Вот некоторые полезные методы (таблица 4.1) класса Assert.

Таблица 4.1 – Описание методов класса Assert

Метод	Описания
void AreEqual(object arg1,object arg2)	если arg1 = arg2 то тест пройден
void AreNotEqual(object arg1,object arg2)	если arg1!=arg2 то тест пройден
void IsNull(object arg)	если arg имеет значение null то тест пройден
void IsNotNull(object arg)	если arg имеет значение null то тест пройден
void IsTrue(bool arg)	если arg имеет значение true то тест пройден
void IsNotTrue(bool arg)	если arg имеет значение false то тест пройден
void Grater(IComparabel arg1,IComparable arg2)	если arg1>arg2 то тест пройден
void Less(IComparable arg1,IComparable arg2)	если arg1<arg2 то тест пройден

Если ожидается, что в методе будет ошибка, то метод следует пометить атрибутом [ExpectedException(“Exception”)] В круглых скобках следует писать тип ожидаемой ошибки.

Создание юнит-тестов в Visual Studio

Создаём новый проект на языке C# как в предыдущем шаге. Затем добавляем в этот проект 2 ссылки. Добавлять ссылки можно, нажав меню Projrct-> Add reference…

Необходимо добавить ссылку на nunit.framework (рисунок 4.2) и на созданный проект (рисунок 4.3)

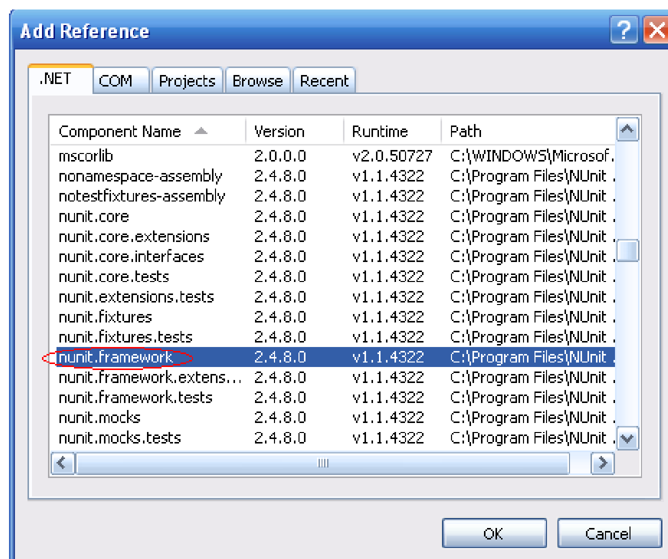


Рисунок 4.2 - Добавление ссылки на nunit.framework

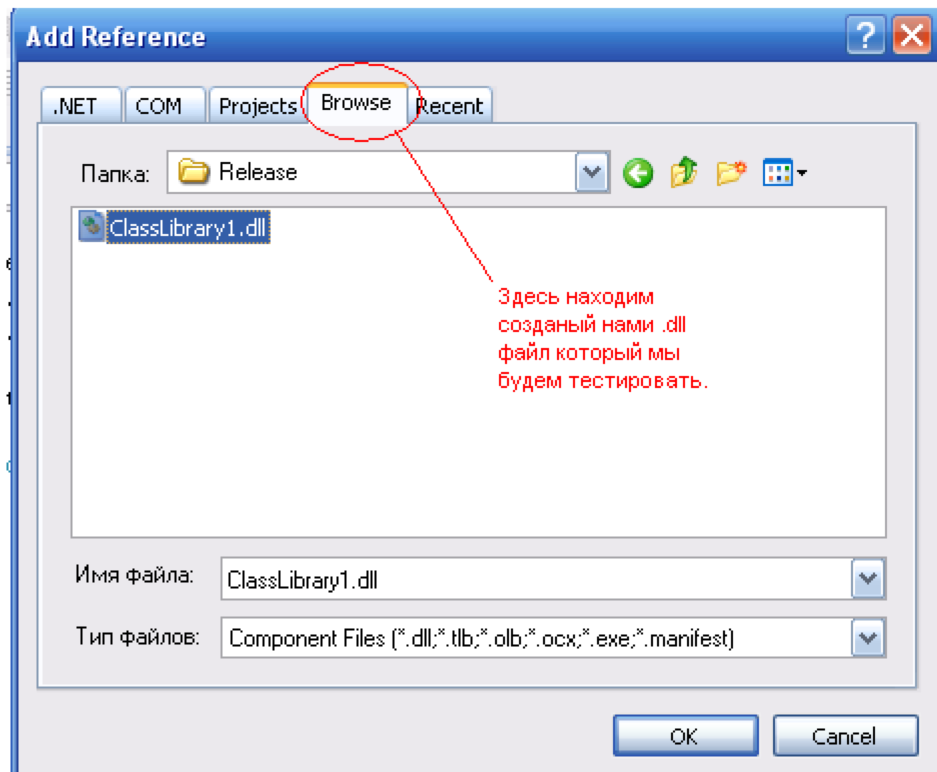


Рисунок 4.3 - Добавление ссылки на проект

Замечания

При компиляции юнит тестов необходимо проверить версию .NET framework рассматриваемого проекта и утилиты NUnit. Если в данном проекте используется версия framework выше чем в NUnit, то его следует поменять. Поменять используемый .NET Framework можно в опциях проета (рисунок 4. 4).

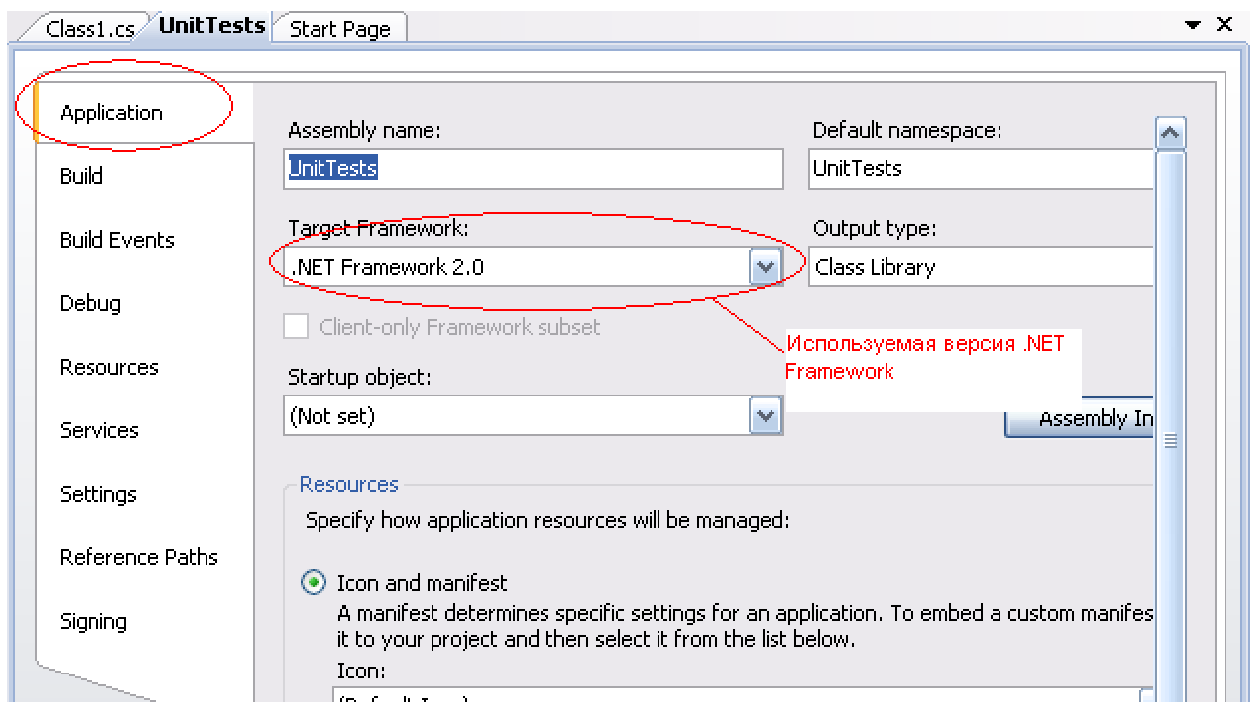


Рисунок 4.4 – Смена .NET framework

Листинг

```
using System;
using System.Collections.Generic;
using System.Text;
using NUnit.Framework;
using ClassLibrary1;//ссылка на тестируемый класс.
namespace UnitTests
{
    [TestFixture]
    public class Class2
    {
        [TestFixture]
        public void ConstrucnorTest()
        {
            Class1 c = new Class1();
            Assert.IsNotNull(c);

        }
        [Test]
        public void ComonWork()
        {
            int[] array={0,1,2,3,4,5,6,7,8,9,10};
            int[] checkArray = null;
            Class1 c = new Class1();
            checkArray = c.RepartArray(array, 1, 6, 2);


            int[] etalon = { 0, 6, 7, 3, 4, 5, 1, 2,8, 9, 10 };

            for (int i = 0; i < etalon.Length; i++)
            {
                Assert.AreEqual(etalon[i], checkArray[i]);
                Console.WriteLine("etalon: {0} Myarray: {1}", etalon[i],
checkArray[i]);
            }
        }

        [Test]
        [ExpectedException("System.ArgumentException")]
        public void CrossedArrays()
        {
            int[] array = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
            int[] checkArray = null;
            Class1 c = new Class1();
            checkArray = c.RepartArray(array, 1, 2, 2);
            int[] etalon = { 0, 6, 7, 3, 4, 5, 1, 2, 9, 10 };

        }
        [Test]
        [ExpectedException("System.NullReferenceException")]
        public void CachNullReferenceExceptio()
        {
            int[] array = null;
            int[] checkArray = null;
            Class1 c = new Class1();
            checkArray = c.RepartArray(array, 1, 2, 2);
        }
        [Test]
        public void WrongTest()
        {
            Assert.AreEqual(0, 1);

        }

    }
}
```

Теперь подробно разберём каждый из этих тестов.

Тест 1 ConstructorTest

```
[TestFixture]
        public void ConstrucnorTest()
        {
            Class1 c = new Class1();
            Assert.IsNotNull(c);

        }
```

В данном тесте проверяется возвращает ли коструктор класса объект или нет.

Тест 2 ComonWork

```
[Test]
        public void ComonWork()
        {
            int[] array={0,1,2,3,4,5,6,7,8,9,10};
            int[] checkArray = null;
            Class1 c = new Class1();
            checkArray = c.RepartArray(array, 1, 6, 2);


            int[] etalon = { 0, 6, 7, 3, 4, 5, 1, 2,8, 9, 10 };

            for (int i = 0; i < etalon.Length; i++)
            {
                Assert.AreEqual(etalon[i], checkArray[i]);
                Console.WriteLine("etalon: {0} Myarray: {1}", etalon[i],
checkArray[i]);
            }
        }
```

В данном тесте проверяется работа испытуемого класса в обычном режиме работы. Объявляются 2 массива. Один с исходными данными – это массив array, и один с ожидаемым результатом-массив etalon. Ещё один массив получаем, вызвав метод RepartMassiv с исходными данными - это массив checkArray, затем сравниваем ожидаемый результат с полученным в цикле.

Тест 3 CrossedArrays()

```
[Test]
        [ExpectedException("System.ArgumentException")]
        public void CrossedArrays()
        {
            int[] array = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
            int[] checkArray = null;
            Class1 c = new Class1();
            checkArray = c.RepartArray(array, 1, 2, 2);
            int[] etalon = { 0, 6, 7, 3, 4, 5, 1, 2, 9, 10 };

        }
```

В данном тесте ожидается ошибка, поскольку в метод RepartArray были введены неверные индексы. Если ошибки не будет, то тест не пройден.

Тест 4 CachNullReferenceExceptio

```
[Test]
        [ExpectedException("System.NullReferenceException")]
        public void CachNullReferenceExceptio()
        {
         int[] array = null;
         int[] checkArray = null;
         Class1 c = new Class1();
         checkArray = c.RepartArray(array, 1, 2, 2);
    }
```

В данном тесте ожидается ошибка NullReferenceException.

Тест 5 WrongTest()

```
[Test]
        public void WrongTest()
        {
            Assert.AreEqual(0, 1);

        }
```

Данный тест неправельный и написан преднамеренно, чтобы показать ошибку в тесте.

Как видно из листинга, возможности тестирования здесь не ограничены. Можно извлекать входные данные из файла, можно выводить даные в консоль, в trace в файл и т. д.

После написания юнит тестов проект следует откомпилировать в .dll файл.

Шаг 3. Прогон тестов в визуальной среде

Для начала следует открыть утилиту NUnit. Затем необходимо открыть .dll файл с юнит тестами. Сделать это можно, нажав File->Open Project. В случае удачи в рабочем поле вкладки Test появятся заголовки написаных тестов (рисунок 4.5).

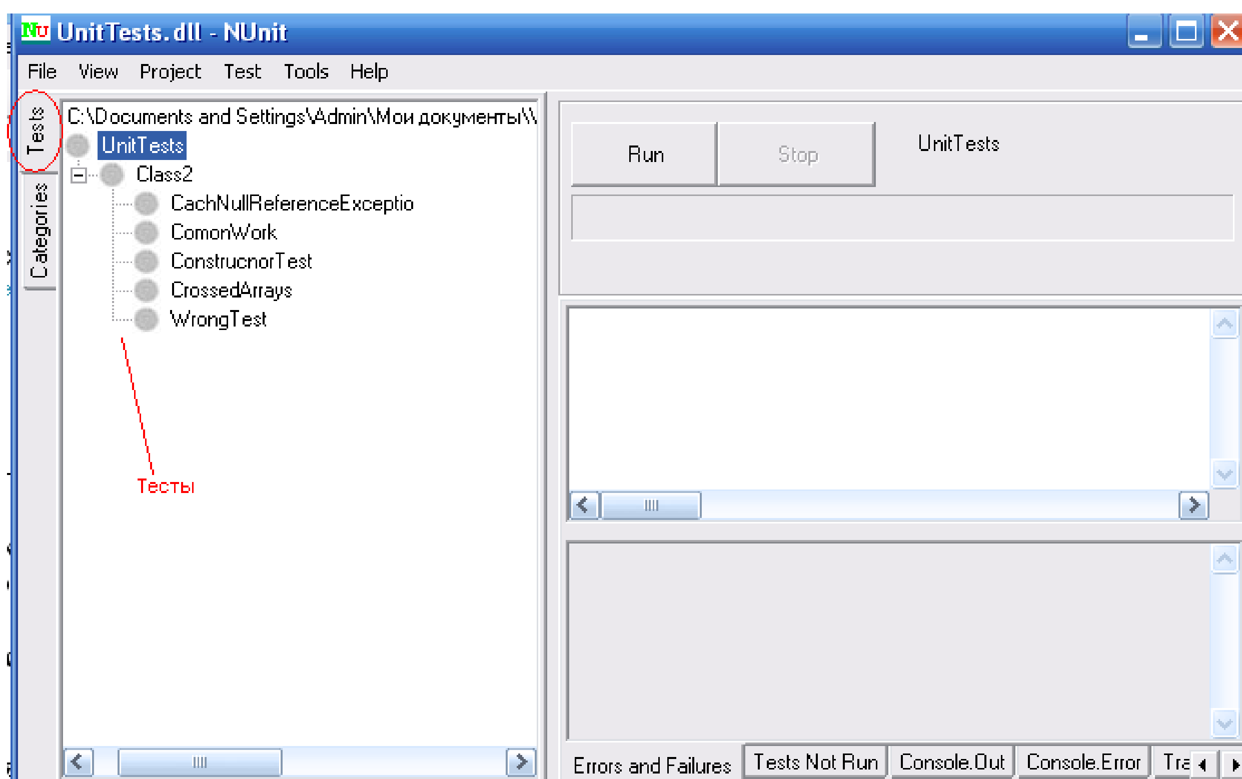


Рисунок 4.5 – Визуальная среда NUnit

Для запуска тестов необходимо нажать кнопку Run. Далее ожидает следующая кортина (рисунок 4.6).

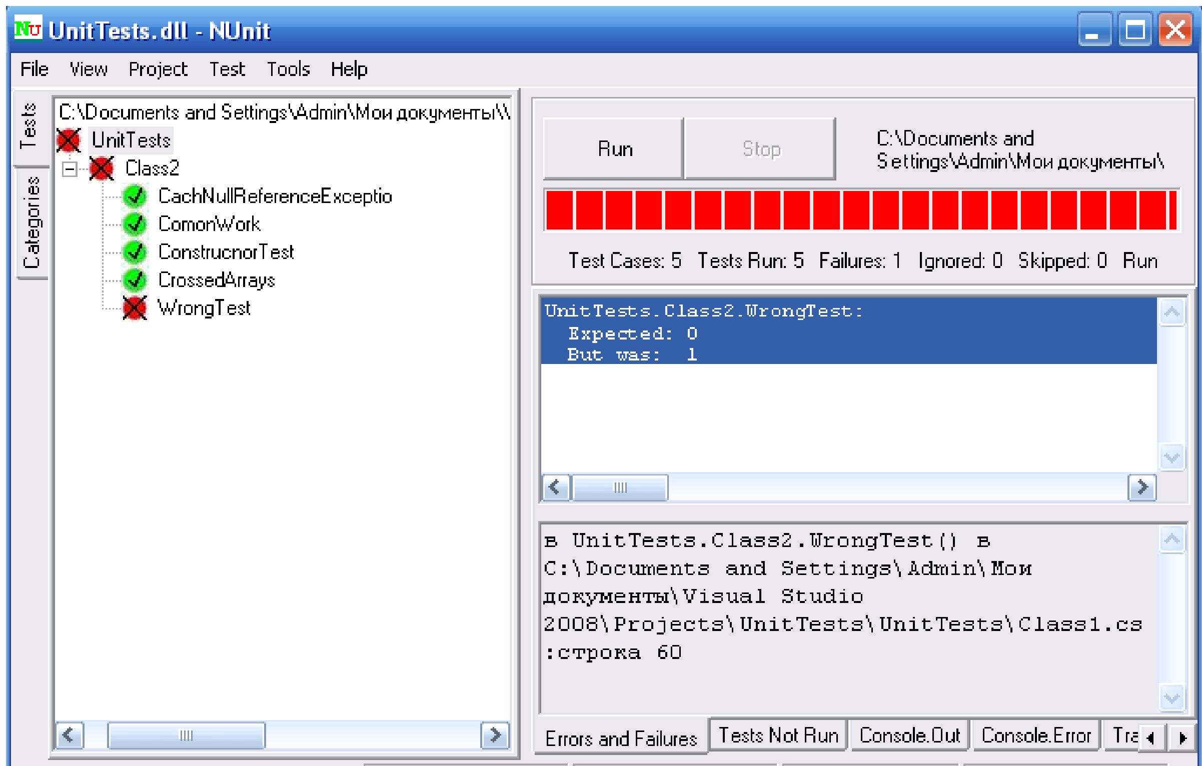


Рисунок 4.6 – Запуск тестов

Тесты, помеченые зелёным цветом, пройдены, те что красным – не прошли.

Во вкладке Errors and Failures указываются ошибки непройденых тестов. Во вкладке Console.Out можно увидеть вывод в консоль, который был получен в тесте ComonWork (рисунок 4.7).

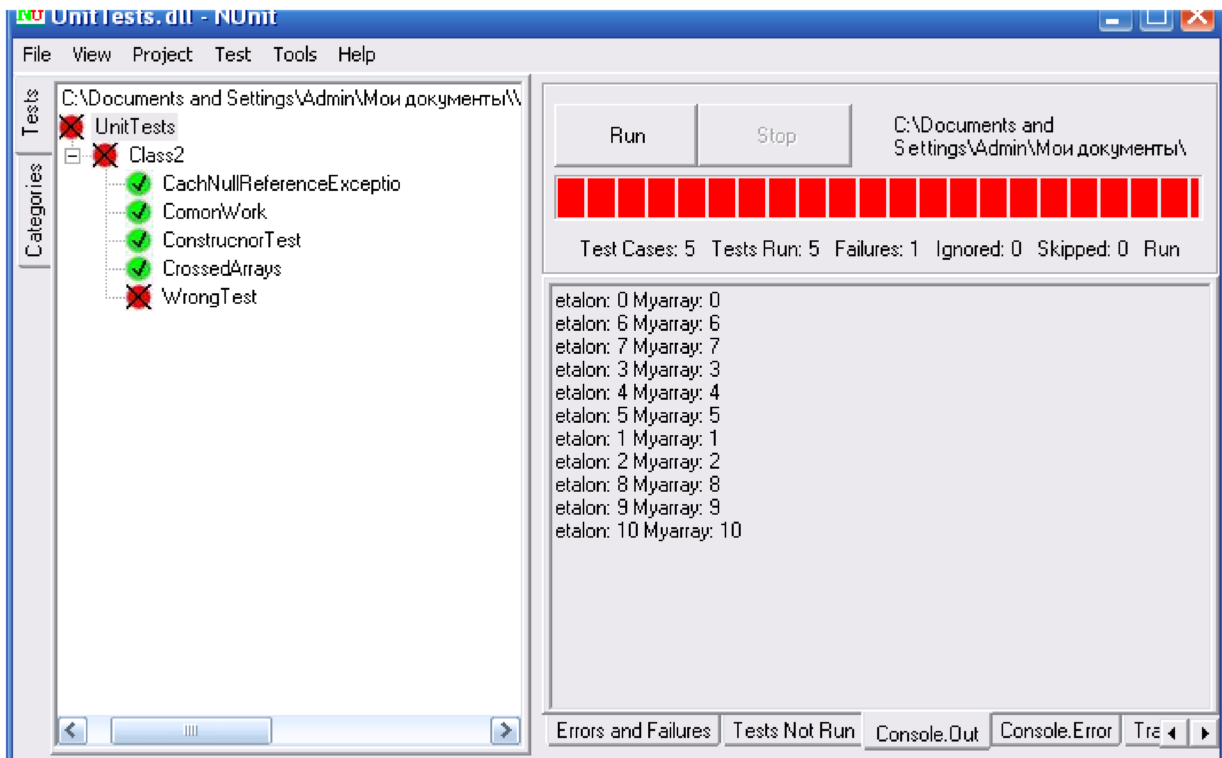


Рисунок 4.7 – Консольный вывод в NUnit

Запускать тесты можно по одному, а можно и все сразу.

Заключение

Утилита NUnit очень удобна при работе группы программистов. Каждый рядовой программист при написании своего модуля пишет юнит тесты, которые являются свидетельством корректной работы модуля. Главный программист при приёме отдельного модуля запускает юнит тесты, и если они все прошли, то данный модуль считается протестированным. При этом рядовой программист не должен показывать в ручную каждый тест, экономя время себе и главному программисту. Всё происходит автоматически.

NUnit используется в связке с утилитами NCover. При использовании NUnit с NCover можно узнать процент охваченого тестами кода. В идеале все тесты должны обеспечивать покрытие кода в 100 процентов. На практике же является обязательным покрытие кода более 90 процентов. В Интернете пожно найти плагины, которые интегрируют NCover и NUnit с Visual Studio.

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: Автоматическое тестирование Selenium

Цель работы состоит в приобретении навыков по**:**

- изучению инструмента для автоматизированного управления Selenium;
- созданию тестовых вариантов, используя фреймворк Selenium, реализуемых на C#, Python;
- повышению навыков программирования на C#, Python;
- отображению результатов тестирования на веб-странице.

В процессе выполнения работы студенты закрепляют свои теоретические знания, касающиеся автоматического тестирования, DOM структуры HTML страницы и работы с библиотеками автоматического тестирования.

Указания и предложения по работе

Selenium — это инструмент для автоматизированного управления браузерами. Наиболее популярной областью применения **Selenium** является автоматизация тестирования веб-приложений. Однако при помощи **Selenium** можно (и даже нужно!) автоматизировать любые другие рутинные действия, выполняемые через браузер.

Разработка **Selenium** поддерживается производителями популярных браузеров. Они адаптируют браузеры для более тесной интеграции с **Selenium**, а иногда даже реализуют встроенную поддержку **Selenium** в браузере.

Selenium является центральным компонентом целого ряда других инструментов и фреймворков автоматизации.

Selenium поддерживает десктопные и мобильные браузеры.

Selenium позволяет разрабатывать сценарии автоматизации практически на любом языке программирования. С помощью **Selenium** можно организовывать распределённые стенды, состоящие из сотен машин с разными операционными системами и браузерами, и даже выполнять сценарии в облаках.

Selenium — это настоящее и будущее автоматизированного управления браузерами. Профессионал в области автоматизации тестирования веб-приложений обязательно должен владеть этим инструментом.

Для выполнения данной лабораторной работы понадобится **Selenium WebDriver,** набор библиотек для различных языков программирования, позволяющих управлять браузером из программы, написанной на этом языке программирования.

Преимущества Selenium:

- Selenium - инструмент с открытым исходным кодом.
- Имеет возможности выполнять скрипты в разных браузерах.
- Может выполнять скрипты в различных операционных системах.
- Поддерживает мобильные устройства.
- Выполняет тесты в браузере, поэтому фокус НЕ требуется во время выполнения сценария.
- Может выполнять тесты параллельно с использованием селеновых сеток.

Для начала работы необходимо настроить среду программирования, для этого понадобятся:

- Текстовый редактор или IDE (в нашем случае будем использовать Visual Studio Code).
- Версия Python 3.6.4.
- Библиотека Selenium для Python.
- Chrome Driver.
- Visual Studio.

В данной лабораторной работе будут описаны примеры четырёх тестируемых функционалов:

- Открыть сайт https://google.com и сделать автоматический поиск по запросу в поле поисковика.
- Открыть сайт https://translate.google.com и произвести автоматический перевод с языка на язык, озвучить переведённую фразу, поменять языки местами.
- Открыть сайт https://utm.md, автоматизировать переход между разделами сайта.
- Открыть сайт https://aliexpress.com и выполнить поиск товара по запросу с определенными фильтрами поиска.

После установки всех необходимых пунктов, описанных выше, выполняем следующие шаги:

Шаг 1. Создаем проектную папку (например, на рабочем столе)

Шаг 2. Открываем **Visutal Studio Code,** выбираем опцию **Open folder…** и выбираем папку, созданную в пункте 1 (рисунок 5.1).

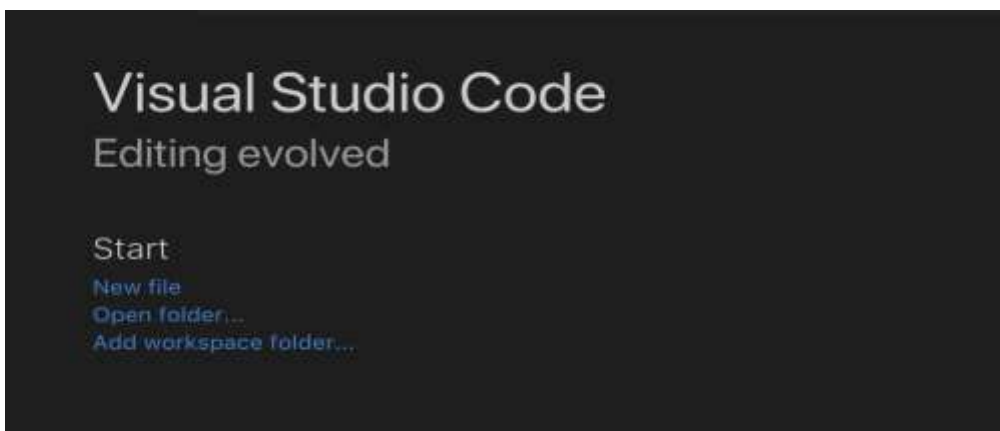


Рисунок 5.1 – Выбор рабочей папки

Шаг 3. Создаем новый файл, нажав на кнопку **New File (**рисунок 5.2) (в данном случае — это 1.py).

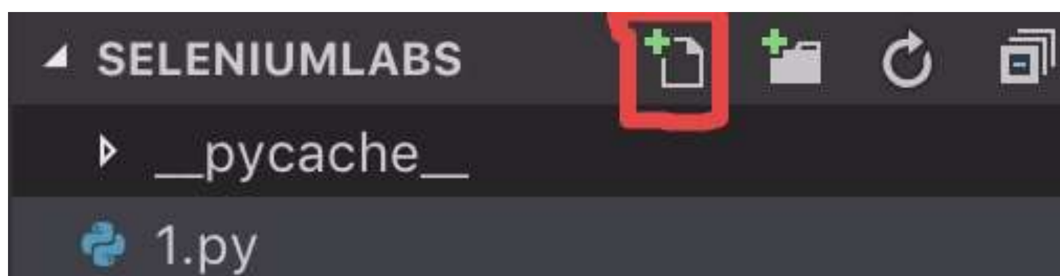


Рисунок 5.2 – Создание файла

Шаг 4. Открываем командную строку и вбиваем команду (при условии, что установлен Python, в противном случае скачиваем по ссылке https://www.python.org/downloads/)**:**

pip install selenium

pip install webdriver_manager

Шаг 5. Создаём файл и подключаем необходимые библиотеки для работы с автоматизированным тестированием:

```python
import time

# Для работы с webdriver
from selenium import webdriver
# Для использования специальных условий для нахождения элеменов
from selenium.webdriver.support import expected_conditions
# Для использования функционала ожидания до нахождения элемента
from selenium.webdriver.support.wait import WebDriverWait
# Для конкретного списка способов поиска элементов
from selenium.webdriver.common.by import By
# Для эмулирования нажатий клавиш
from selenium.webdriver.common.keys import Keys
# Для использования драйвера Chromium
from webdriver_manager.chrome import ChromeDriverManager
```

Шаг 6. Настраиваем класс для автоматического запуска тестов на последней версии chromium драйвера. При инициализации класса (setup_class) настраиваем переменную драйвера, которую будем использовать для навигации в браузере. Для удобства используем maximize_window, чтобы развернуть браузер на весь экран и настраиваем переменную web_driver_wait для того, чтобы искать элементы в течение 10 секунд. При закрытии класса (teardown_class) драйвер, а соответственно, браузер будет закрыт.

```python
class TestClass:
    new *
    def setup_class(self):
        options = webdriver.ChromeOptions()
        options.add_experimental_option('excludeSwitches', ['enable-logging'])
        self.driver = webdriver.Chrome(ChromeDriverManager().install(), options=options)  # noqa
        self.driver.maximize_window()
        self.web_driver_wait = WebDriverWait(self.driver, 10)  # noqa

    new *
    def teardown_class(self):
        self.driver.quit()
```

Шаг 7. Ниже показано тело первой программы с подробными комментариями.

Для поиска элементов Selenium предлагает множество вариантов условий:

```
search_field = self.web_driver_wait.until(
    expected_conditions.visibility_of_element_located((By.ID, 'APjFqb')),
    "Can't find searc
)
# search_field = self
# search_field = self

# Передаём принудител
# Может сработать и:
# search_field.click(
self.driver.execute_s

# В имеющийся элемент
# search_field.send_k
search_field.send_key
# search_field.send_k
# search_field.send_k

# Даём нам 10 секунд
time.sleep(10)
```

```
visibility_of_element_located    selenium.webdriver.suppor…
element_to_be_clickable    selenium.webdriver.support.expe…
presence_of_element_located    selenium.webdriver.support.…
url_changes    selenium.webdriver.support.expected_conditi…
text_to_be_present_in_element    selenium.webdriver.suppor…
visibility_of    selenium.webdriver.support.expected_condi…
alert_is_present    selenium.webdriver.support.expected_co…
all_of(e…    selenium.webdriver.support.expected_conditions
any_of(e…    selenium.webdriver.support.expected_conditions
element_attribute_to_include    selenium.webdriver.support…
element_located_selection_state_to_be    selenium.webdrive…
element_located_to_be_selected    selenium.webdriver.suppo…
element_selection_state_to_be    selenium.webdriver.suppor…
element_to_be_selected    selenium.webdriver.support.expec…
frame_to_be_available_and_switch_to_it    selenium.webdriv…
invisibility_of_element    selenium.webdriver.support.expe…
invisibility_of_element_located    selenium.webdriver.supp…
new_window_is_opened    selenium.webdriver.support.expecte…
NoAlertPresentException    selenium.common.exceptions
Press Enter to insert, Tab to replace  Next Tip
```

Для запуска программы используем консоль или функционал среды разработки. Совет: для запуска можно запускать тесты не просто через python, а используя pytest.

```
def test_google_search(self):
    google_url = 'https://google.com/'
    request = 'Selenium'

    # Открываем страницу поисковика
    self.driver.get(google_url)

    # By.ID, By.CSS_SELECTOR, By.NAME, By.CLASS_NAME, By.TAG_NAME, By.LINK_TEXT, By.PARTIAL_LINK_TEXT, By.XPATH
    # В течение 10 секунд проверяем видимость элемента с ID 'APjFqb' на странице и сохраняем элемент
    search_field = self.web_driver_wait.until(
        expected_conditions.visibility_of_element_located((By.ID, 'APjFqb')),
        "Can't find search area"
    )
    # search_field = self.driver.find_element(By.NAME, 'q')
    # search_field = self.driver.find_element_by_name('q')

    # Передаём принудительное нажатие на данное поле.
    # Может сработать и:
    # search_field.click()
    self.driver.execute_script('arguments[0].click();', search_field)

    # В имеющийся элемент передаём текстовые данные и вместо поиска кнопки "Найти" используем "Enter"
    # search_field.send_keys(request + '\n')
    search_field.send_keys(request + Keys.ENTER)
    # search_field.send_keys(request)
    # search_field.send_keys(Keys.ENTER)

    # Даём нам 10 секунд насладиться проделанной работой
    time.sleep(10)
```

Результат выполнения программы:

Результат работы программы представлен на рисунке 5.3.

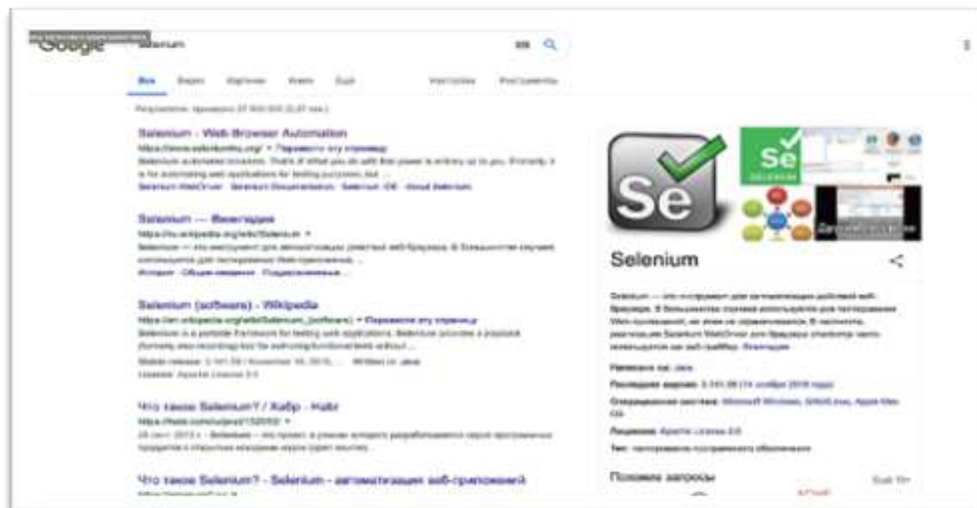


Рисунок 5.3 – Результат работы программы

Тестируемый функционал № 2

Задание: открыть сайт https://translate.google.com и произвести автоматический перевод с языка на язык, озвучить переведённую фразу, поменять языки местами.

Шаги выполнения

Шаг 1. Подключаем необходимые библиотеки (рисунок 5.4).

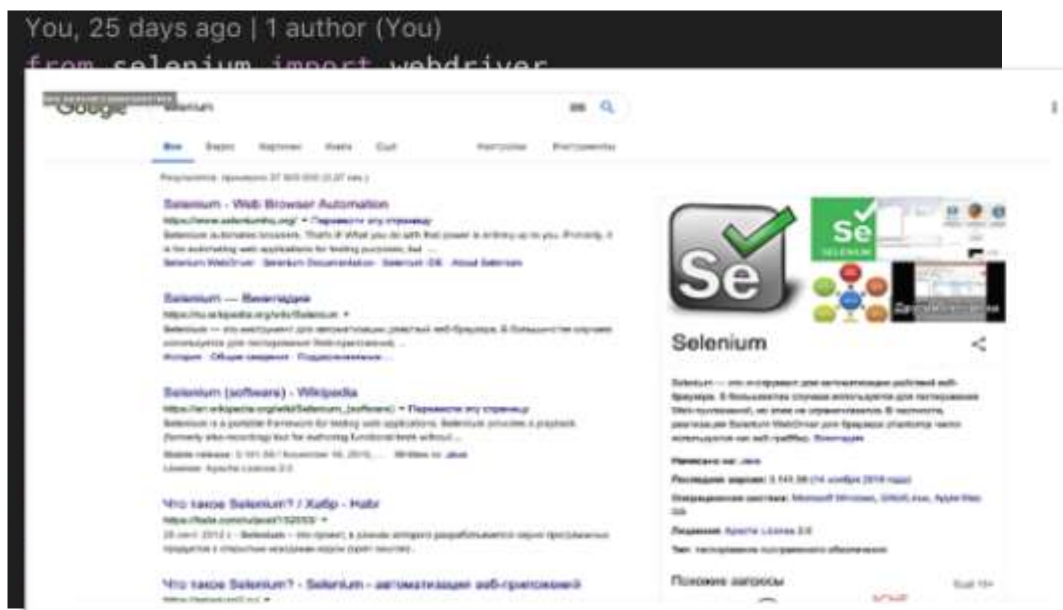


Рисунок 5.4 – Driver.py - подключение библиотек

Шаг 2. После подключения библиотек прописываем код программы (рисунок 5.4).

Запускаем функцию драйвера, объявленную в файле **Driver.py.**

Создаем переменную с адресом сайта.

Создаем переменную текста, который нужно перевести.

вызываем метод за открытие веб-страницы.

находим **HTML-элемент** текстового поля и передаем в него текст**.**

находим **HTML-элемент** кнопки звукового воспроизведения переведённого текста, производим клик на него методом **click().**

находим **HTML-элемент** кнопки для переключения языка, производим клик на него методом **click().**

```
def test_google_translate(self):
    translate_url = 'https://translate.google.com/'
    text_for_translate = 'Hello world'

    self.driver.get(translate_url)
    self.web_driver_wait.until(
        expected_conditions.visibility_of_element_located((By.CLASS_NAME, 'er8xn')),
        "Can't find translate text area"
    ).send_keys(text_for_translate)

    self.driver.execute_script(
        'arguments[0].click();',
        self.web_driver_wait.until(
            expected_conditions.element_to_be_clickable((
                By.XPATH, '//*[@class="VO9ucd"]/div[1]/div[2]/span/button')),
            "Can't find voice read button"
        )
    )
    time.sleep(3)

    self.driver.execute_script(
        'arguments[0].click();',
        self.web_driver_wait.until(
            expected_conditions.visibility_of_element_located((
                By.XPATH, '//*[@class="aCQag"]/c-wiz/div[1]/c-wiz/div[3]/div[1]/span/button')),
            "Can't find switch language button"
        )
    )

    time.sleep(10)
```

Рисунок 5.5 – Driver.py - тело программы

Результат

Результат работы программы представлен на рисунке 5.6.

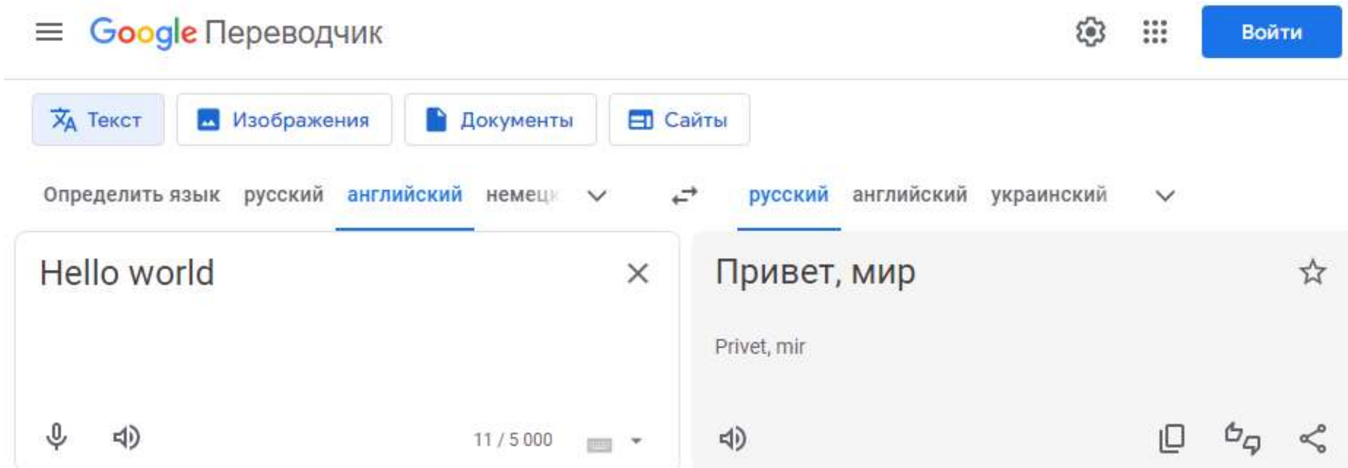
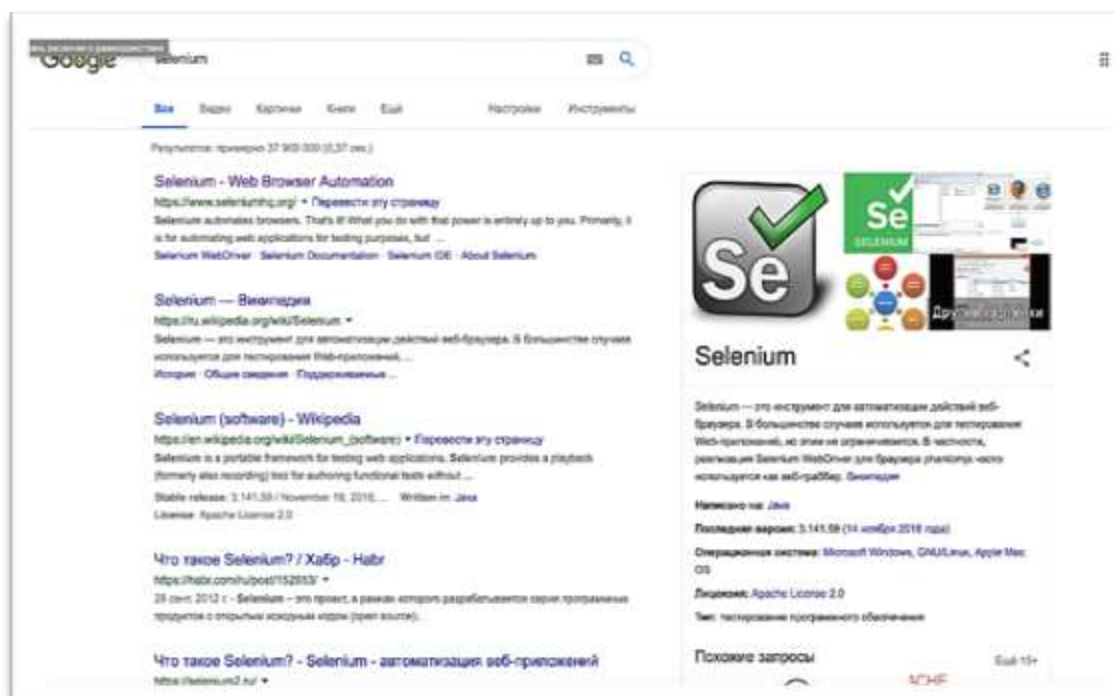


Рисунок 5.6 – Результат работы программы

Тестируемый функционал №3

Задание: открыть сайт https://utm.md, автоматизировать переход между разделами сайта.



Шаги выполнения

Рассмотрим детально тело программы 3-го тестируемого функционала.

Код программы навигации по сайту университета, представлен на рисунке 5.7.

Запускаем функцию драйвера, объявленную в файле **Driver.py.**

Создаем переменную с адресом сайта.

Находим **HTML-элемент** раздела блога университета на сайте и имитируем нажатие левой кнопки мыши с помощью метода **click().**

Проверяем, что переход на страницу блога университета произошёл успешно.

```python
def test_utm_navigate(self):
    utm_url = 'https://utm.md/'

    self.driver.get(utm_url)
    self.driver.execute_script(
        'arguments[0].click();',
        self.web_driver_wait.until(
            expected_conditions.presence_of_element_located((By.CSS_SELECTOR, '[href="https://utm.md/blog"]')),
            "Can't find in menu link to 'https://utm.md/blog'"
        )
    )

    self.web_driver_wait.until(
        expected_conditions.url_changes(utm_url),
        "After click to the link we still on the UTM welcome page"
    )

    time.sleep(1)
    self.driver.execute_script('window.scrollTo(0,1000)')
    self.driver.execute_script("alert('Scroll down finished')")

    time.sleep(10)
```

Рисунок 5.7 – Driver.py - тело программы

Результат

Результаты работы программы представлен на рисунке 5.8.

Рисунок 5.8 – Результаты работы программы

Тестируемый функционал № 4

Задание: открыть сайт https://aliexpress.com и выполнить поиск товара по запросу с определенными фильтрами поиска.

Шаги выполнения

Рассмотрим детально тело программы 4-го тестируемого сайта (рисунок 5.9):

Запускаем функцию драйвера, объявленную в файле **Driver.py.**

Создаем переменную с адресом сайта.

Создаем переменную текста необходимого нам товара.

Переход на веб-страницу.

Находим **HTML-элемент** поля ввода поиска товара на сайте **aliexpress**, после чего передаем в него товар и осуществляем нажатие клавиши **ENTER** с помощью метода **send_keys.**

```
def test_aliexpress_shop(self):
    ali_url = 'https://aliexpress.com/'
    search_text = 'laptop'
    min_price = '5000'
    max_price = '10000'

    self.driver.get(ali_url)
    self.web_driver_wait.until(
        expected_conditions.visibility_of_element_located((By.ID, 'searchInput')),
        "Can't find search area"
    ).send_keys(search_text + Keys.ENTER)

    min_max_fields = self.driver.find_elements(By.CLASS_NAME, 'snow-ali-kit_Input__inputField__1aiyxh')
    min_field = min_max_fields[0]
    max_field = min_max_fields[1]

    min_field.send_keys(min_price)
    max_field.send_keys(max_price)

    self.web_driver_wait.until(
        expected_conditions.visibility_of_element_located((By.ID, 'searchInput')),
        "Can't find search area"
    ).send_keys(Keys.ENTER)

    # Даём нам 10 секунд насладиться проделанной работой
    time.sleep(10)
```

Рисунок 5.9 – Driver.py - тело программы

Находим **HTML-элементы** фильтров, по которому ищем нужный товар (минимальная и максимальная цены). Производим поиск элементов с помощью следующих методов на выбор (рисунок 5.10).

Рисунок 5.10 – Методы для поиска HTML-элементов на странице, предоставляемые библиотекой Selenium

После этого передаем нужные параметры в зависимости от фильтра и эмулируем нажатие клавиши Enter.

Результат

Результат работы программы представлен на рисунке 5.11.

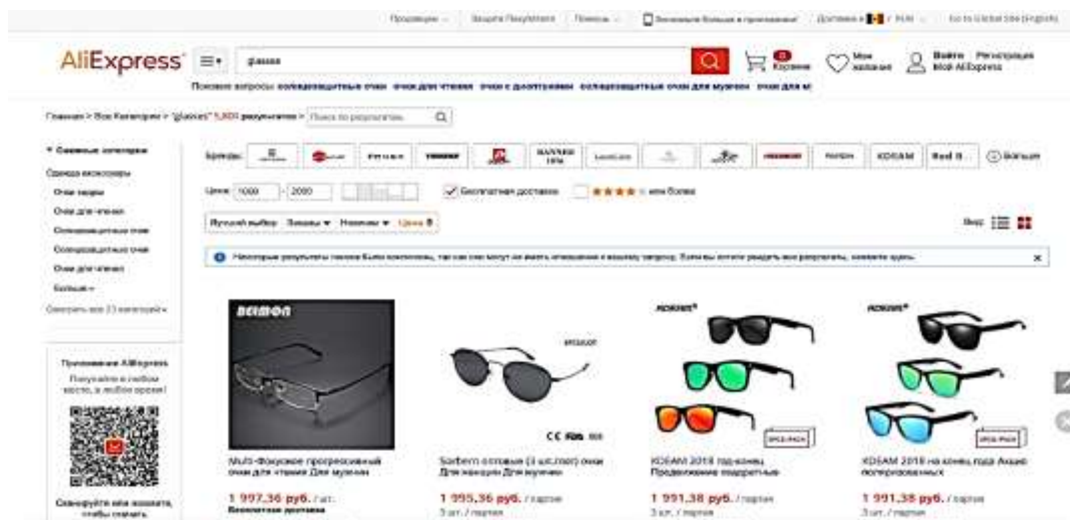


Рисунок 5.11 – Результат работы программы

Контрольные вопросы:

1. Что такое Selenium?
2. Для чего используется **Selenium WebDriver**?
3. Что позволяют делать методы **send_keys(), find_element_by_....?**
4. Перечислите преимущества **Selenium.**
5. Как определяются нужные элементы на странице сайта?

Задания для лабораторной работы

На оценку 8

1. Открыть сайт https://yandex.ru и сделать автоматический поиск по запросу в поле поисковика.
2. Открыть сайт https://translate.yandex.ru и произвести автоматический перевод с языка на язык, озвучить переведённую фразу, поменять языки местами.
3. Открыть сайт https://www.bing.com и сделать автоматический поиск по запросу в поле поисковика.
4. Открыть сайт https://pikabu.ru/ и пролайкать первые 5 постов.
5. Зайти на сайт https://fcim.utm.md/ проскролить вниз до раздела новостей, открыть последнюю новость, кликнуть на фотографию. Если есть несколько фотографий, используя стрелочки прокрутить их.
6. Открыть сайт https://www.translate.ru/dictionary/ и произвести автоматический перевод с языка на язык, озвучить переведённую фразу, поменять языки местами.
7. Авторизироваться на сайте https://mail.ru/.
8. Перейти на сайт https://twitter.com/, зайти в первую актуальную тему и проскролить первые 10 постов.

На оценку 9 – 10

1. Зайти на сайт https://www.booking.com/, найти Кишинев, открыть самый популярный отель, кликнуть на фотографии и в режиме просмотра фотографий, используя стрелочки прокрутить 5 фотографий.
2. Открыть сайт https://translate.yandex.ru и произвести автоматический перевод с языка на язык, озвучить переведённую фразу, поменять языки местами.
3. Открыть сайт https://yandex.ru и сделать автоматический поиск по запросу в поле поисковика.
4. Авторизироваться на сайте https://mail.ru/.
5. Открыть сайт https://pikabu.ru/ и пролайкать первые 10 постов.
6. Авторизироваться на сайте https://www.google.com/ или на любом другом, где есть возможность.
7. Открыть сайт https://www.reddit.com/ выполнить поиск по запросу UTM и прокрутить в самый низ страницы, в конце страницы должен выскочить js alert с надписью "Hello Reddit".
8. Зайти на сайт https://fcim.utm.md/ проскролить вниз до раздела новостей, открыть последнюю новость, кликнуть на фотографию. Если есть несколько фотографий, используя стрелочки прокрутить их.
9. Открыть сайт https://ru.stackoverflow.com зайти на 5 случайных постов и на каждом пробыть минимум 5 секунд.

10. Открыть сайт https://ru.m.wikipedia.org, открыть 5 случайных статей и на каждой должен выскочить js alert с названием статьи.
11. Открыть сайт https://ru.stackoverflow.com зайти на 5 случайных постов и на каждом пробыть минимум 5 секунд.
12. Регистрация на сайте - сайт на ваш выбор. Разрешается полуавтоматическая программа (можно вводить капчу вручную)

ЛАБОРАТОРНАЯ РАБОТА № 6

Тема: Автоматическое тестирование мобильных приложений с помощью инструмента UIAutomator

Цель работы

Приобретение навыков по созданию тестов для автоматизации тестирования мобильных приложений с помощью инструмента UIAutomator.

Задание:

– Установка и настройка среды разработки Android Studio / Eclipse.

– Установка Android SDK.

– Настройка UIAutomatorViewer для определения элементов экрана.

– Изучение функций для поиска и запуска приложения.

– Разработка скрипта для тестирования.

– Сбор логов и отчётов о тестировании.

Указания и предложения по работе

Тестирование — это очень важный процесс во время разработки приложений. В случае Android, тестирование приложения следует производить на большом количестве устройств, в связи с тем, что многие из них имеют значительные различия по характеристикам (разрешение экрана, версия Android и т. д.). Процесс тестирования приложения вручную на большом количестве устройств может быть трудоемок, утомителен и подвержен ошибкам. Более эффективный и надежный подход состоит в автоматизации тестирования пользовательского интерфейса. С помощью UIAutomator можно разработать тест-скрипт, который будет работать на множестве Android устройств с одинаковой точностью и воспроизводимостью.

UIAutomator

UIAutomator разрабатывается корпорацией Google и поставляется вместе с Android SDK. UIAutomator – это аналог инструмента UIAutomation компании Apple для тестирования Android приложений. Android SDK предоставляет следующие инструменты для поддержки автоматизированного функционального тестирования пользовательского интерфейса:

– UIAutomatorviewer — графический инструмент для распознавания компонентов

пользовательского интерфейса в Android приложении;
- UIAutomator — библиотеки Java API, содержащие методы для создания тестов пользовательского интерфейса.

Чтобы использовать эти инструменты, необходимо установить следующие компоненты среды Android:

- Android SDK Tools, версия 21 или выше;
- платформа Android SDK, с API 16 или выше.

UIAutomator постоянно дорабатывается, поэтому самую свежую документацию можно найти на сайте: http://developer.android.com/tools/help/uiautomator/index.html.

Преимущества UIAutomator для тестирования приложений:

- отсутствие зависимости от разрешения экрана;
- действия привязываются к Android UI компонентам. Это позволяет работать напрямую с элементами пользовательского интерфейса. Например, если необходимо нажать кнопку «OK», можно средствами UIAutomator API отправить скрипту команду: нажми кнопку с надписью «OK», и он её нажимает. Таким образом, не приходится привязываться к координатам;
- можно воспроизводить сложные последовательности действий пользователя, и всегда эта последовательность будет одинаковой;
- тесты могут быть запущены необходимое количество раз на различных устройствах без необходимости изменения Java кода;
- можно использовать внешние кнопки на устройстве (кнопка «назад», «выключить», «громкость» и т. д.).

Недостатки:

- тяжело использовать для приложений, написанных на HTML 5 и OpenGL, так как в этих приложениях нет Android UI элементов. В связи с этим, необходимо либо привязываться к координатам, либо искать альтернативные варианты тестирования;
- необходимо проверять, и в случае необходимости, обновлять Java скрипты при обновлении Android приложения.

Тестирование приложения с помощью UIAutomator состоит из следующих шагов:

Шаг 1. Подготовка к тесту: установка приложения на устройство, анализ его UI компонент.

Шаг 2. Создание автоматизированного теста для приложения.

Шаг 3. Компиляция теста в JAR файл и копирование его на устройство.

Шаг 4. Запуск теста и анализ результатов.

Шаг 5. Исправление различных ошибок, найденных в процессе тестирования.

Разработка скрипта

Для ознакомления с технологиями UIAutomator далее будет представлена простая программа, осуществляющая несложные действия с устройством. В качестве тестируемого приложения будет использовано стандартное Android приложение — Messaging, а UIAutomator будет отправлять SMS-сообщение на определенный номер.

Определим действия, которые будут реализованы в тесте:

- поиск и запуск приложения;
- создание и отправка сообщения.

Подготовка к тесту

Для анализа пользовательского интерфейса приложения будет использоваться UIAutomatorviewer. UIAutomatorviewer делает снимок экрана устройства (показано на рисунке 6.1), который подключен к компьютеру, а также предоставляет удобный графический интерфейс для отображения иерархии слоев и просмотра свойств каждого компонента интерфейса в отдельности. Наличие этой информации значительно упрощает процесс создания UIAutomator скрипта.

Рисунок 6.1 – Скриншот с изображением UIAutomatorviewer

Для анализа пользовательского интерфейса необходимо:

- присоединить Android устройство к компьютеру;
- запустить UIAutomatorviewer, находящийся в: <android-sdk>/tools/ (рисунок 6.2);
- $ uiautomatorviewer;

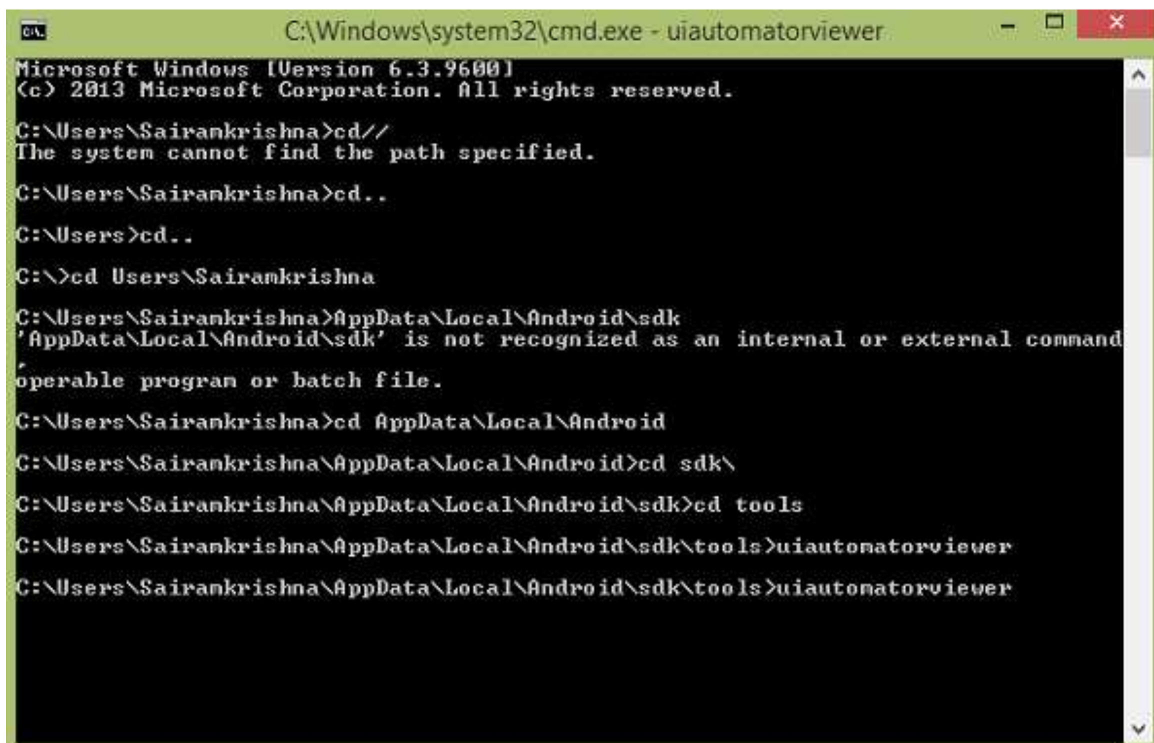


Рисунок 6.2 – Скриншот запуска из терминала

- нажать на кнопку Device Screenshot в UIAutomatorviewer для захвата изображения с устройства (рисунок 6.3);

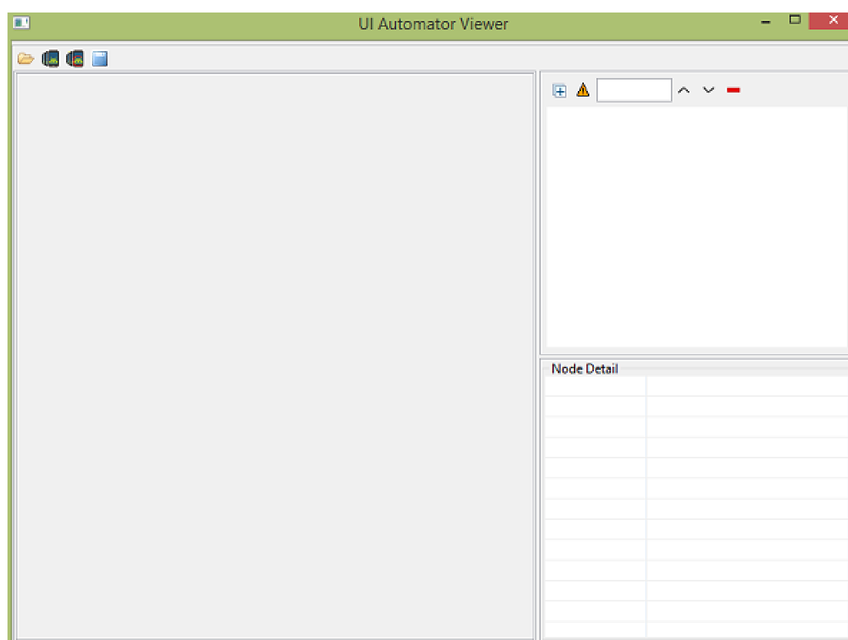


Рисунок 6.3 – Скриншот запущенного UIAutomatorviewer

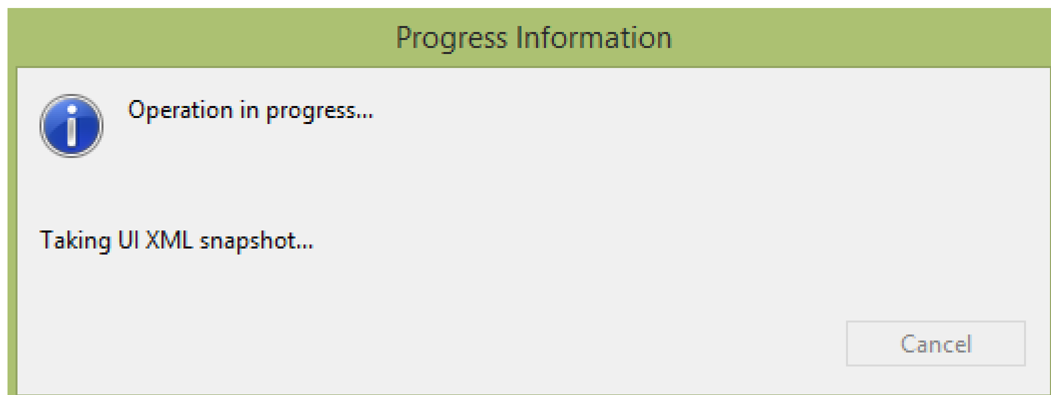


Рисунок 6.4 – Процесс создания скриншота экрана

– при выделении определенного элемента интерфейса на панели справа отображаются его свойства. По этим свойствам элемент легко может быть найден в тест-скрипте (рисунок 6.5).

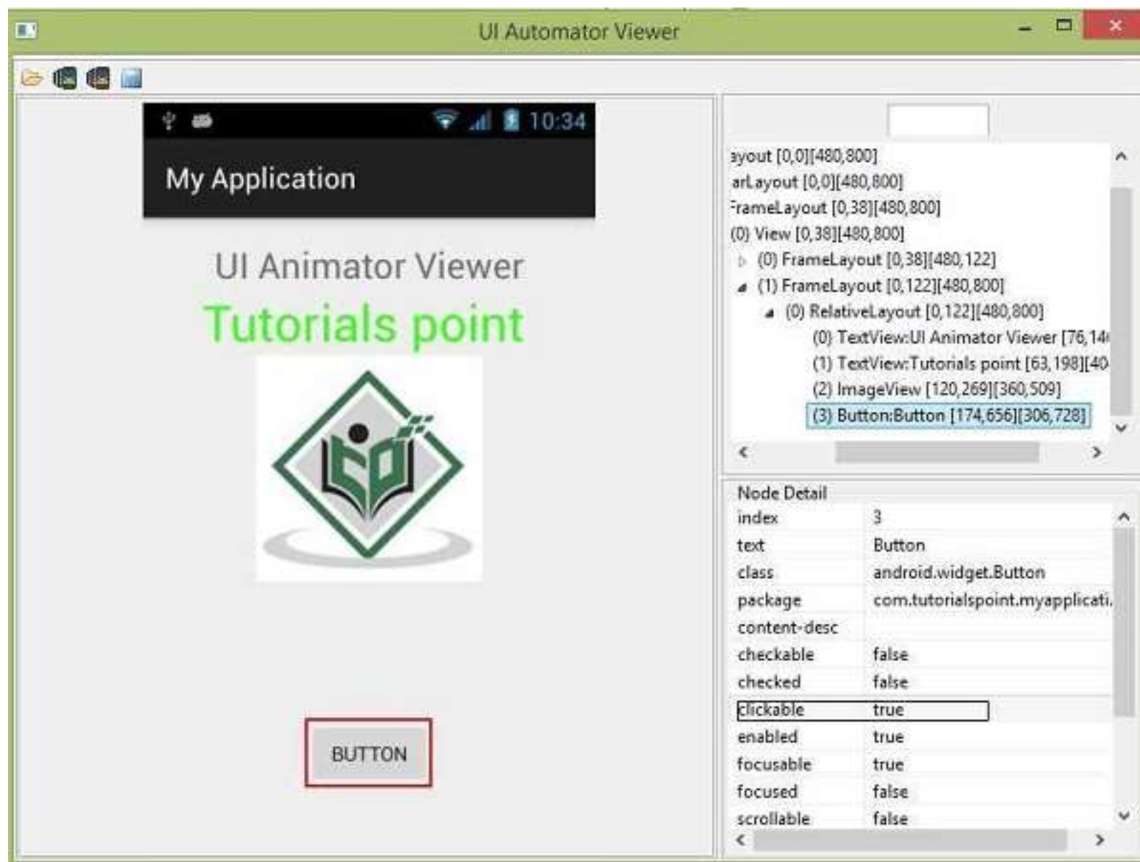


Рисунок 6.5 – Скриншот загруженного изображения телефона

Если UIAutomatorviewer не может разложить изображение на компоненты, значит, приложение написано с использованием HTML 5 или OpenGL.

Настройка среды разработки

Если используется Eclipse:

a) Создаем новый Java проект в Eclipse. В этом проекте будем реализовывать UIAutomator скрипт. Назовем проект: SendMessage.
b) Вызываем контекстное меню, нажав правой кнопкой мыши на проект в Project Explorer, и выбираем пункт Properties. В Properties находим Java Build Path и добавляем необходимые библиотеки:
   1) для добавления поддержки JUnit необходимо выбрать Add Library > JUnit;
   2) нажав Add External JARs, необходимо перейти в директорию с последней версией SDK в <android-sdk>/platforms/ и выбрать в ней 2 файла: uiautomator.jar и android.jar.

При использовании другой среды разработки, убедитесь, что файлы UIAutomator.jar и android.jar добавлены в настройки проекта. Если вы выбрали среду разработки Android Studio, то лучше использовать Espresso, вместо UIAutomator, на сегодняшний день он является более предпочтительным для Android Studio https://developer.android.com/training/testing/espresso

UIAutomator API

Чтобы рассказать обо всех возможностях, которыми обладает UIAutomator для создания тест-скриптов, потребуется достаточно большое количество времени. Всю подробную информацию можно найти на сайте: http://developer.android.com/tools/help/UIAutomator/index.html.

Создание скрипта

Для начала необходимо создать в проекте «SendMessage» новый файл с Java классом, например, под именем SendMessage. Этот класс должен быть унаследован от класса UIAutomatorTestCase. Чтобы добавить библиотеку в Eclipse, достаточно воспользоваться сочетанием клавиш Ctrl+Shift+o. Аналогичным образом добавляются и другие библиотеки.

Создадим три функции для тестирования основного функционала приложения:

- Поиск и запуск приложения.
- Отправка SMS сообщения.
- Выход в главное меню приложения.

Первая функция, запускающая все эти методы, своего рода main функция, будет выглядеть следующим образом:

```
public void test() {
// Here will be called for all other functions }
```

Функция для поиска и запуска приложения

Функция будет осуществлять следующие действия: нажать на кнопку Home, для перехода в главном окно, открыть меню и найти значок с нужным приложением, запустить это приложение.

Поиск и запуск приложения

```
private void findAndRunApp() throws UiObjectNotFoundException {
                 // Go to main screen
                 getUiDevice().pressHome();
                 // Find menu button
                 UiObject allAppsButton = new UiObject(new UiSelector()
                 .description("Apps"));
                 // Click on menu button and wait new window
                 allAppsButton.clickAndWaitForNewWindow();
                 // Find App tab
                 UiObject appsTab = new UiObject(new UiSelector()
                 .text("Apps"));
                 // Click on app tab
                 appsTab.click();
                 // Find scroll object (menu scroll)
                 UiScrollable appViews = new UiScrollable(new UiSelector()
                 .scrollable(true));
                 // Set the swiping mode to horizontal (the default is vertical)
                 appViews.setAsHorizontalList();
                 // Find Messaging application
                 UiObject settingsApp = appViews.getChildByText(new UiSelector()
                 .className("android.widget.TextView"), "Messaging");
                 // Open Messaging application
                 settingsApp.clickAndWaitForNewWindow();

                 // Validate that the package name is the expected one
            UiObject settingsValidation = new UiObject(new UiSelector()
            .packageName("com.android.mms"));
            assertTrue("Unable to detect Messaging",
                         settingsValidation.exists());
         }
```

Все названия классов, текста на кнопках и т. д. были получены с помощью uiautomatorviewer.

Отправка SMS сообщения

В этой функции реализуем поиск и нажатие на кнопку создания нового сообщения, ввод номера телефона, текста сообщения и его последующая отправка. Номер и текст будут передаваться через аргументы функции:

Функция отправки sms

```
private void sendMessage(String toNumber, String text) throws UiObjectNotFoundException {
```

```
        // Find and click New message button
        UiObject newMessageButton = new UiObject(new UiSelector()
        .className("android.widget.TextView").description("New message"));
        newMessageButton.clickAndWaitForNewWindow();

        // Find to box and enter the number into it
        UiObject toBox = new UiObject(new UiSelector()
        .className("android.widget.MultiAutoCompleteTextView").instance(0));
        toBox.setText(toNumber);
        // Find text box and enter the message into it
        UiObject textBox = new UiObject(new UiSelector()
        .className("android.widget.EditText").instance(0));
        textBox.setText(text);

        // Find send button and send message
        UiObject sendButton = new UiObject(new UiSelector()
        .className("android.widget.ImageButton").description("Send"));
        sendButton.click();
    }
```

Поле для номера и поле для сообщения не смогли бы найти по каким-либо особым признакам, поскольку ни текста, никакого-либо описания у этих форм нет. Поэтому находим их с помощью метода instance. Этим методом можно получить элемент по его порядковому номеру в иерархии интерфейса.

Реализуем возможность получения в качестве параметров номера телефона получателя, а также текст сообщения. В функцию test() необходимо добавить инициализацию параметров по умолчанию, которые должны быть перезаписаны пользовательскими значениями, если они были переданы в качестве аргументов соответствующей функции.

Прием параметров в тесте

```
    // Default parameters
        String toNumber = "123456";
        String text = "Test message";

        String toParam = getParams().getString("to");
        String textParam = getParams().getString("text");
    if (toParam != null) {
        // Remove spaces
            toNumber = toParam.trim();
        }
        if (textParam != null) {
```

```
                        text = textParam.trim();
                }
```

Таким образом, можно передавать параметры через командную строчку в скрипт. Это можно делать с помощью ключа – e. После него передается 2 значения: имя параметра и значение. Например, для передачи номера «777777», в качестве номера получателя, передадим параметры: - e to 777777. Для получения этих параметров в скрипте используется метод: getParams().

Но тут есть и несколько подводных камней. Например, не получается передать текст с некоторыми символами, UIAutomator их не воспринимает (пробел, &, <, >, (,), “, ’, и т.д., а также юникодные символы). Для этого предлагается делать замену этих символов при подаче их в скрипт какой-нибудь строкой, например, пробел заменить строкой: blogspaceblog. Это удобно, когда для запуска скрипта UIAutomator используется скрипт, который будет обрабатывать входные параметры. Добавим в проверку входных параметров парсинг и замену этих строк:

Код замены строк

```
        if (toParam != null) {
                        toNumber = toParam.trim();
                }
                if (textParam != null) {
                        textParam = textParam.replace("blogspaceblog", " ");
                        textParam = textParam.replace("blogamperblog", "&");
                        textParam = textParam.replace("bloglessblog", "<");
                        textParam = textParam.replace("blogmoreblog", ">");
                        textParam = textParam.replace("blogopenbktblog", "(");
                        textParam = textParam.replace("blogclosebktblog", ")");
                        textParam = textParam.replace("blogonequoteblog", "'");
                        textParam = textParam.replace("blogtwicequoteblog", "\"");
                        text = textParam.trim();
                }
```

Выход в главное меню приложения

Эта функция самая простая из всех тех, которые были реализованы до этого. Она нажимает кнопку назад, до тех пор, пока не найдет кнопку для создания нового сообщения.

```
private void exitToMainWindow() {
                // Find New message button
                UiObject newMessageButton = new UiObject(new UiSelector()
                .className("android.widget.TextView").description("New message"));

                // Press back button while new message button doesn't exist
                while(!newMessageButton.exists()) {
                        getUiDevice().pressBack();
                }
        }
```

Сбор логов с теста

Для того, чтобы логировать результаты теста, можно использовать стандартный буфер Android. Для работы с ним необходимо подключить библиотеку в скрипте:

```
import android.util.Log;
```

Всю информацию, которая интересна можно записывать в логи. Это можно делать с помощью функции:

```
Log.i(String title, String title);
```

Логин можно читать с устройства с помощью команды:

```
$ adb logcat
```

Более подробную информацию по logcat можно найти на официальном сайте: developer.android.com/tools/help/logcat.html

Получившийся код

Таким образом, получен такой код:

```
package blog.send.message;
import android.util.Log;
import com.android.UIAutomator.core.UiObject;
import com.android.UIAutomator.core.UiObjectNotFoundException;
import com.android.UIAutomator.core.UiScrollable;
import com.android.UIAutomator.core.UiSelector;
import com.android.UIAutomator.testrunner.UiAutomatorTestCase;

public class SendMessage extends UiAutomatorTestCase {
        public void test() throws UiObjectNotFoundException {
                // Default parameters
                String toNumber = "123456";
                String text = "Test message";

                String toParam = getParams().getString("to");
                String textParam = getParams().getString("text");
                if (toParam != null) {
                        toNumber = toParam.trim();
                }
                if (textParam != null) {
                        textParam = textParam.replace("blogspaceblog", " ");
                        textParam = textParam.replace("blogamperblog", "&");
                        textParam = textParam.replace("bloglessblog", "<");
                        textParam = textParam.replace("blogmoreblog", ">");
                        textParam = textParam.replace("blogopenbktblog", "(");
                        textParam = textParam.replace("blogclosebktblog", ")");
                        textParam = textParam.replace("blogonequoteblog", "'");
                        textParam = textParam.replace("blogtwicequoteblog", "\"");
                        text = textParam.trim();
                }
```

```
                Log.i("SendMessageTest", "Start SendMessage");
                findAndRunApp();
                sendMessage(toNumber, text);

                exitToMainWindow();
    3.              Log.i("SendMessageTest", "End SendMessage");
        }
        // Here will be called for all other functions
        private void findAndRunApp() throws UiObjectNotFoundException {
                // Go to main screen
                getUiDevice().pressHome();
                // Find menu button
                UiObject allAppsButton = new UiObject(new UiSelector()
                .description("Apps"));
                // Click on menu button and wait new window
                allAppsButton.clickAndWaitForNewWindow();
                // Find App tab
                UiObject appsTab = new UiObject(new UiSelector()
                .text("Apps"));
                // Click on app tab
                appsTab.click();
                // Find scroll object (menu scroll)
                UiScrollable appViews = new UiScrollable(new UiSelector()
                .scrollable(true));
                // Set the swiping mode to horizontal (the default is vertical)
                appViews.setAsHorizontalList();
                // Find Messaging application
                UiObject settingsApp = appViews.getChildByText(new UiSelector()
                .className("android.widget.TextView"), "Messaging");
                // Open Messaging application
                settingsApp.clickAndWaitForNewWindow();

                // Validate that the package name is the expected one
            UiObject settingsValidation = new UiObject(new UiSelector()
            .packageName("com.android.mms"));
            assertTrue("Unable to detect Messaging",
                        settingsValidation.exists());
        }

        private void sendMessage(String toNumber, String text) throws
UiObjectNotFoundException {
                // Find and click New message button
                UiObject newMessageButton = new UiObject(new UiSelector()
                .className("android.widget.TextView").description("New message"));
                newMessageButton.clickAndWaitForNewWindow();

                // Find to box and enter the number into it
```

```
                UiObject toBox = new UiObject(new UiSelector()
                .className("android.widget.MultiAutoCompleteTextView").instance(0));
                toBox.setText(toNumber);
                // Find text box and enter the message into it
                UiObject textBox = new UiObject(new UiSelector()
                .className("android.widget.EditText").instance(0));
                textBox.setText(text);

                // Find send button and send message
                UiObject sendButton = new UiObject(new UiSelector()
                .className("android.widget.ImageButton").description("Send"));
                sendButton.click();
        }

        private void exitToMainWindow() {
                // Find New message button
                UiObject newMessageButton = new UiObject(new UiSelector()
                .className("android.widget.TextView").description("New message"));

                // Press back button while new message button doesn't exist
                while(!newMessageButton.exists()) {
                        getUiDevice().pressBack();
                        sleep(500);
                }
        }
}
```

Компиляция и запуск UIAutomator теста

Для генерации файлов конфигурации сборки теста необходимо выполнить следующую команду в терминале:

```
$ <android-sdk>/tools/android create uitest-project -n <name> -t  <target-id> -p <path>
```

где <name>- имя проекта, который создавался для теста UIAutomator (в данном случае: SendMessage),

<target-id> - выбор устройства и Android API Level (можно получить список установленных устройств командой: <android-sdk>/tools/android list targets) и - путь к директории с проектом.

Необходимо экспортировать переменную окружения ANDROID_HOME:

– Windows:

```
set ANDROID_HOME=<path_to_your_sdk>
```

– Unix:

```
export ANDROID_HOME=<path_to_your_sdk>
```

Заходим в директорию с проектом, в которой лежит сгенерированный на шаге 1 файл build.xml и выполняем команду:

```
$ ant build
```

Копируем собранный JAR файл на устройство с помощью команды adb push:

```
$ adb push <path_to_output_jar> /data/local/tmp/
```

Для данного случая:

```
$ adb push <project_dir>/bin/SendMessage.jar /data/local/tmp/
```

Запускаем скрипт:

```
$ adb shell uiautomator runtest /data/local/tmp/SendMessage.jar -c blog.send.message.SendMessage -e to 777777
```

При использовании Android Studio запускаем каждый класс теста, используя список классов для тестирования.

Клик правой кнопкой мыши на класс → Run «TestCaseName»

Примеры реализаций тестов

Java

Тест case запуск и запись камеры:

```
package com.example.testing;

import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.os.RemoteException;
import android.support.test.InstrumentationRegistry;
import android.support.test.runner.AndroidJUnit4;

import org.junit.Before;
```

```
import org.junit.Test;
import org.junit.runner.RunWith;

import java.util.concurrent.TimeUnit;

import androidx.test.uiautomator.By;
import androidx.test.uiautomator.UiDevice;
import androidx.test.uiautomator.UiObject;
import androidx.test.uiautomator.UiObjectNotFoundException;
import androidx.test.uiautomator.UiSelector;
import androidx.test.uiautomator.Until;

@RunWith(AndroidJUnit4.class)
public class TestCaseCamera {

    private UiDevice mDevice;
    @Before
    public void setUp() throws RemoteException, UiObjectNotFoundException {
        mDevice = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());
        mDevice.pressHome();
        mDevice.wait(Until.hasObject(By.pkg(getLauncherPackageName()).depth(0)), 1000);
    }

    @Test
    public void checkSettings() throws UiObjectNotFoundException, InterruptedException,
RemoteException {
        mDevice.pressHome();

        UiObject camera = mDevice.findObject(new UiSelector().text("Camera"));
        camera.clickAndWaitForNewWindow();
        camera.waitUntilGone(300);

        UiObject video = mDevice.findObject(new
UiSelector().resourceId("com.android.camera:id/v6_module_picker"));
        video.click();

        UiObject start = mDevice.findObject(new
UiSelector().resourceId("com.android.camera:id/v6_shutter_button_internal"));
        start.click();
        TimeUnit.SECONDS.sleep(3);

        UiObject stop = mDevice.findObject(new
UiSelector().resourceId("com.android.camera:id/v6_shutter_button_internal"));
        stop.click();
        stop.waitUntilGone(300);
```

```
        UiObject gallery = mDevice.findObject(new
UiSelector().resourceId("com.android.camera:id/v6_thumbnail_image_cover"));
        gallery.clickAndWaitForNewWindow();

        UiObject startVideo = mDevice.findObject(new
UiSelector().resourceId("com.miui.gallery:id/video_icon"));
        startVideo.click();
        TimeUnit.SECONDS.sleep(3);

        mDevice.pressBack();
    }

    private String getLauncherPackageName() {
        final Intent intent = new Intent(Intent.ACTION_MAIN);
        intent.addCategory(Intent.CATEGORY_HOME);

        PackageManager pm = InstrumentationRegistry.getContext().getPackageManager();
        ResolveInfo resolveInfo = pm.resolveActivity(intent,
PackageManager.MATCH_DEFAULT_ONLY);
        return resolveInfo.activityInfo.packageName;
    }
}
```

Вывод при успешном (рисунок 6.6) и неуспешном (рисунок 6.7) выполнении теста приведены ниже.

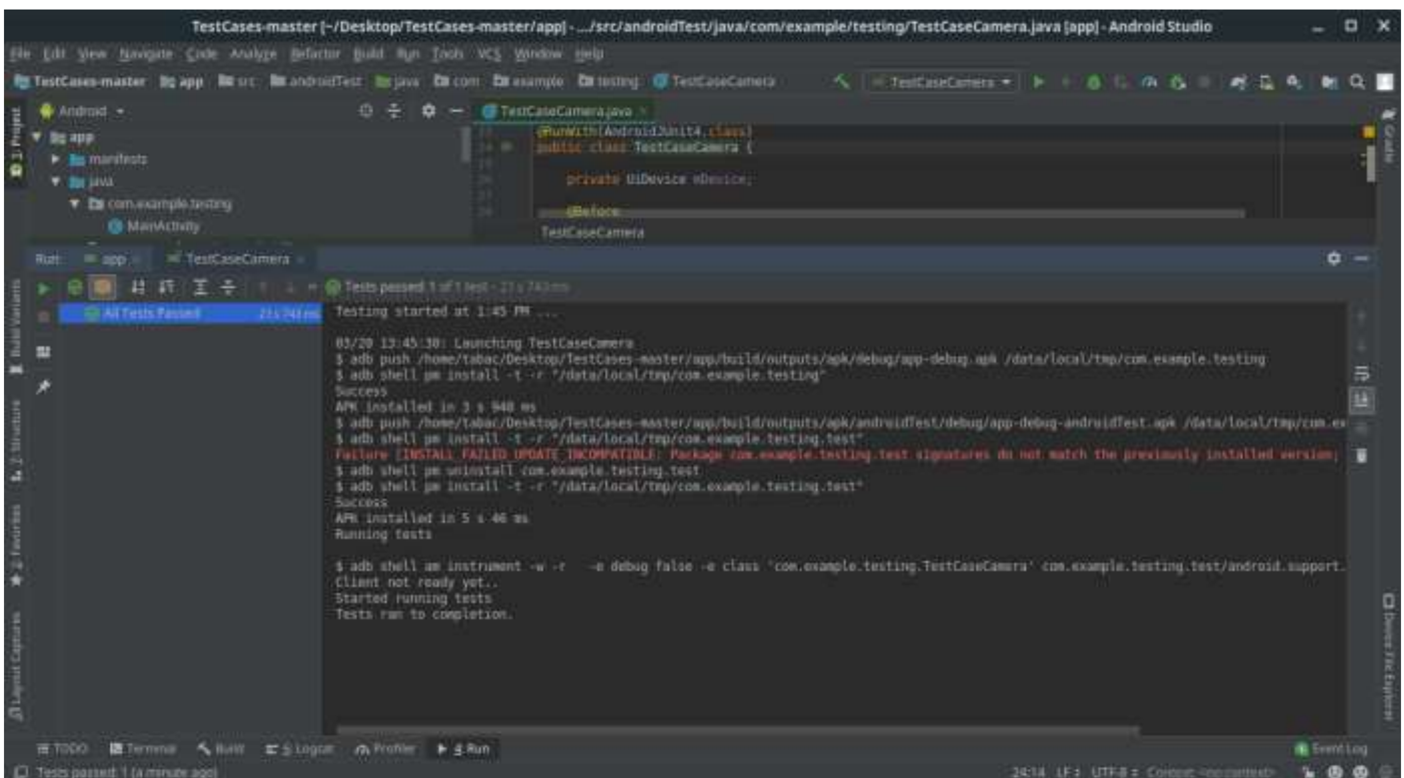


Рисунок 6.6 – Успешное выполнение теста

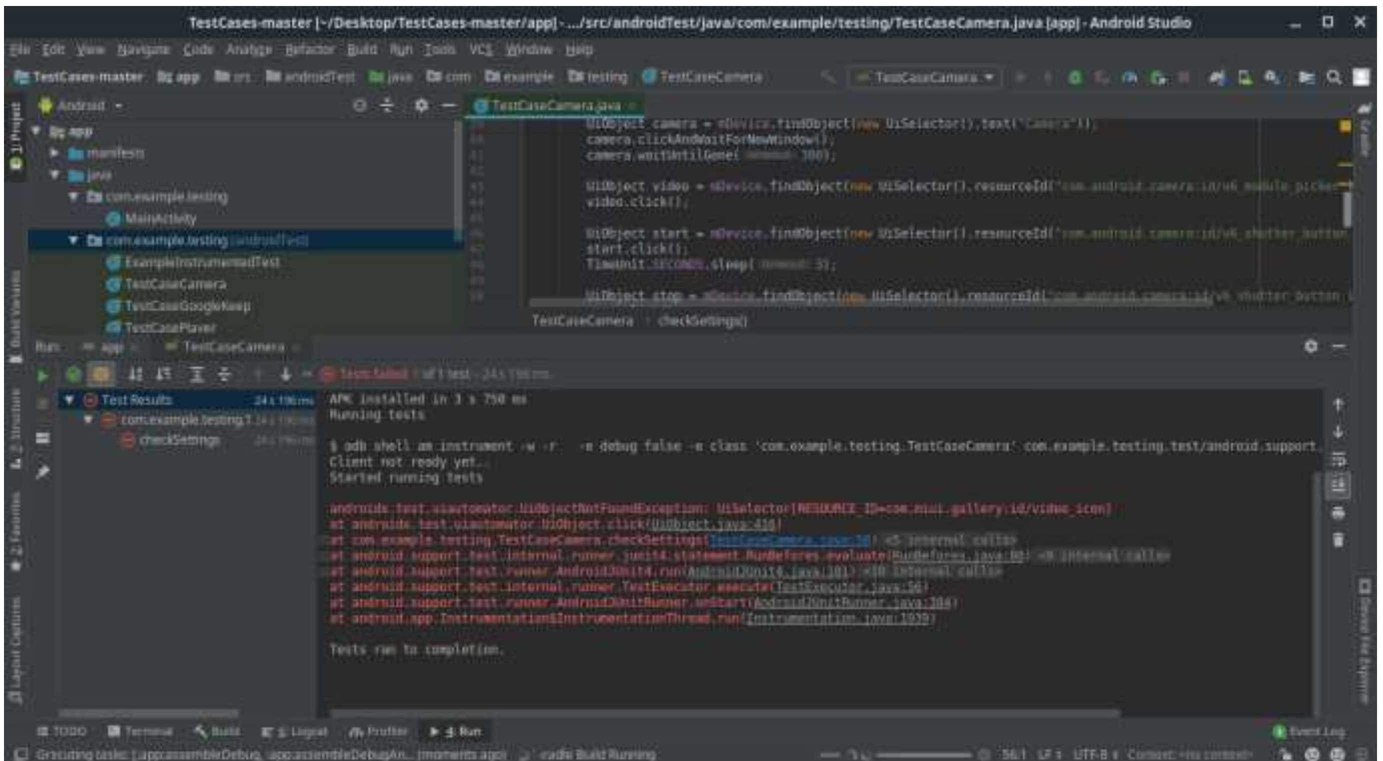


Рисунок 6.7 - Ошибка при выполнении теста: Нужный элемент не найден на экране

Kotlin

Навигация в университет Google maps:

```
package com.github.tigr.uiautomatortests

import android.content.Context
import android.content.Intent
import android.support.test.InstrumentationRegistry
import android.support.test.InstrumentationRegistry.getInstrumentation
import android.support.test.runner.AndroidJUnit4
import android.support.test.uiautomator.By
import android.support.test.uiautomator.UiDevice
import android.support.test.uiautomator.UiSelector
import android.support.test.uiautomator.Until
import org.hamcrest.CoreMatchers.notNullValue

import org.junit.Test
import org.junit.runner.RunWith

import org.junit.Assert.*
import org.junit.Before
import java.lang.Thread.sleep

/**
```

```
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class UiAutomatorTests {
    lateinit var device: UiDevice

    @Before
    fun init() {
        device = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation())
        assertThat(device, notNullValue())

        device.pressHome()
    }

    @Test
    fun findUniversityInGoogleMaps() {
        val launcherPackage = device.launcherPackageName

        device.wait(
                Until.hasObject(By.pkg(launcherPackage).depth(0)),
                5000
        )

        val context =
getInstrumentation().context//ApplicationProvider.getApplicationContext<Context>()
        val intent = context.packageManager.getLaunchIntentForPackage(
                "com.google.android.apps.maps"
        )!!.apply{
            addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
        }
        context.startActivity(intent)

        device.wait(
                Until.hasObject(By.pkg("com.google.android.apps.maps").depth(0)),
                5000
        )

        val searchBar = device.findObject(
            UiSelector()
                .resourceId("com.google.android.apps.maps:id/search_omnibox_text_box")
                .className("android.widget.EditText")
        )
        assertNotNull(searchBar)
        assert(searchBar.isClickable)
        searchBar.click()
```

```
        device.findObject(UiSelector().text("Search here")).setText("Universitatea Tehnica a
Moldovei")
        device.findObject(UiSelector().textContains("Studentilor")).click()
    }
```

Заключение

Использование UIAutomator очень удобно для качественного тестирования приложений на большом количестве девайсов. С помощью этой технологии можно создавать различные Demo для демонстрации приложения.

Варианты заданий для лабораторной работы № 6

Написать тест, согласно варианту:

1. Перевод слова в Google Translate.
2. Добавление нового напоминания Google Keep.
3. Записать видео Camera.
4. Поиск и запуск видео YouTube.
5. Запуск и переключение музыки Player.
6. Навигация в университет Google Maps.
7. Звонок другу Facebook Messenger.
8. Скачать приложение Google Play.
9. Переход на сайт Google Chrome.
10. Поиск девушки в Tinder.
11. Звонок другу Skype.
12. Сохранение картинки из Telegram.
13. Отправка почты Gmail.
14. Создание пустого файла Google Drive.
15. Поиск по хаштегу Instagram.
16. Бронь отеля в приложении Booking.
17. Покупка товара Aliexpress app.
18. Наложение фильтра Snapseed.
19. Добавление новой активности Google Fit.
20. Установка будильника.

ЛАБОРАТОРНАЯ РАБОТА № 7

Тема: Проведение нагрузочного тестирования (среда JMeter).

Цель работы: проведение нагрузочного тестирования приложения в среде JMeter.

Для любого программного приложения, предназначенного для массового обслуживания пользователей, необходимо проводить нагрузочное тестирование на предмет его надежности и отказоустойчивости. JMeter является очень мощным инструментом нагрузочного тестирования с возможностью создания большого количества запросов одновременно, благодаря параллельной работе на нескольких компьютерах. Поддерживает плагины, при помощи которых можно расширить функционал.

Нагрузочное тестирование (load testing) — данный тип тестирования позволяет оценить поведение системы при возрастающей нагрузке, целью нагрузочного тестирования является также определение максимальной нагрузки, которую может выдержать система. Рассмотрим его подробнее: в роли нагрузки может выступать количество пользователей, а также количество операций на сервере.

Производительность при этом определяется следующими факторами:

a) скоростью работы программного обеспечения;
b) скоростью работы аппаратного обеспечения;
c) скоростью работы сети.

Во время тестирования могут осуществляться следующие операции, позволяющие более точно измерить производительность и определить «узкое место» системы:

a) измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;
b) определение количества пользователей, одновременно работающих с приложением;
c) определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций).

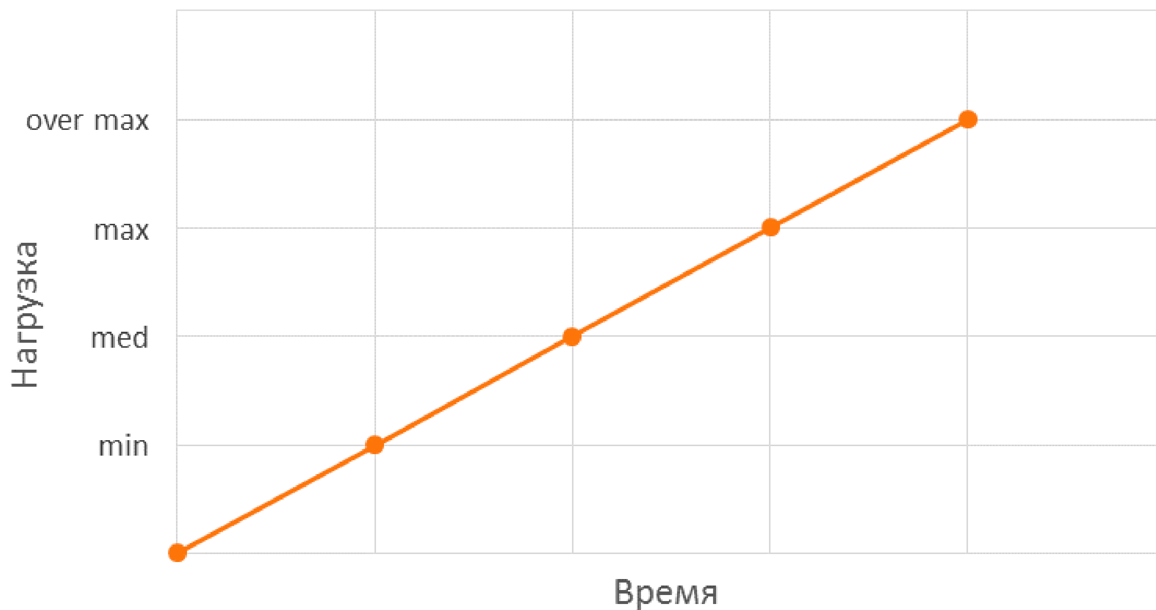


Рисунок 7.1 – Нахождение максимальной производительности

После нахождения максимальной производительности (см. рисунок 7.1) рекомендуется её «подтвердить» (см. рисунок 7.2). Для этого проводится дополнительный тест со следующим профилем:

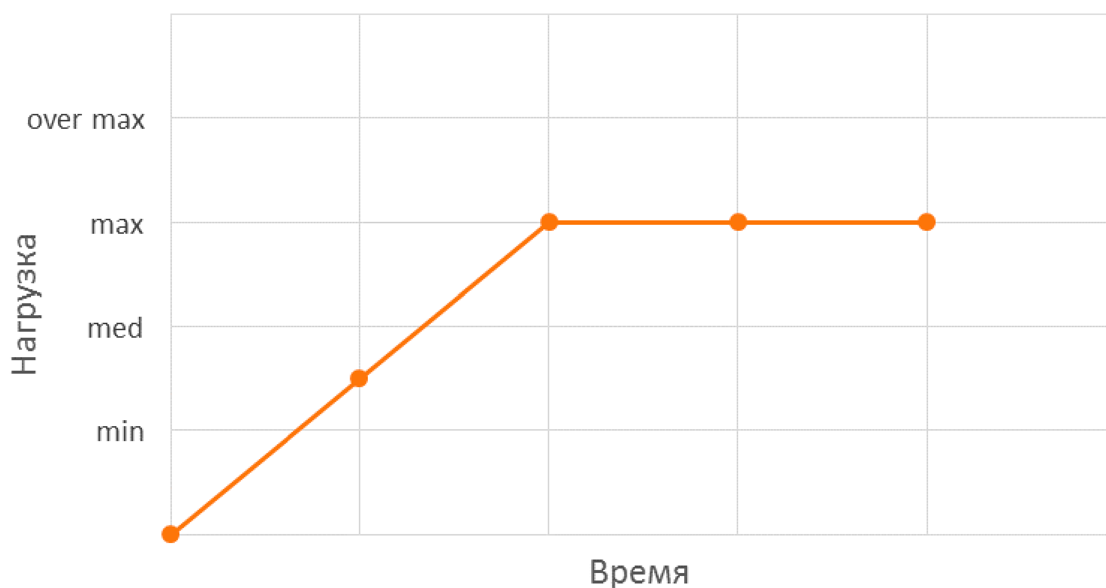


Рисунок 7.2 – «Подтверждение» максимальной производительности

Ход работы

Установка MongoDB

Существует два варианта установки MongoDB: нативно и с помощью Docker

a) Гайд по установке нативно на различные ОС
https://docs.mongodb.com/manual/administration/install-community/

b) Docker:
   1) docker run -p 27017:27017 -d mongo;
   2) Done.

Установка приложения

В данном случае так же существует два варианта развёртывания приложения: с помощью Node.js и с помощью Docker. В любом случае сначала необходимо клонировать репозиторию приложения **https://github.com/bunnies-group/notable-node-api-tutorial**

a) Docker:
   1) docker build -t notable-api .
   2) docker run -p 8000:8000 -e MONGO_URL=$MONGO_URL -d notable-api;

b) Node.js:
   1) установка Node.js:
      – для всех платформ **https://nodejs.org/en/download/**
      – для Linux и Mac также рекомендуется использовать nvm https://github.com/creationix/nvm/blob/master/README.md
      – в случае Mac также можно воспользоваться brew.
   2) запуск приложения происходит с помощью команды MONGO_URL=$MONGO_URL node server.js;

где MONGO_URL – адрес базы данных. Адрес имеет вид mongodb://$IP_ADDRESS:27017;

IP_ADDRESS – это IP-адрес устройства, на котором запущена база данных, в локальной сети.

О том, как определить IP-адрес в локальной сети можно узнать подробнее по ссылке: https://nickjanetakis.com/blog/docker-tip-35-connect-to-a-database-running-on-your-docker-host.

Установка JMeter:

a) Под Mac:
   1) Если есть homebrew -> `brew install jmeter ` в терминале. Если нет, то установить.
   2) Если есть JDK или JRE(лучше), то в терминале: `jmeter` и откроется окно ниже. Если нет -> https://www.java.com/en/download/mac_download.jsp и потом пункт 2 заново

Готово!

b) Под Windows:

1) Установить JDK или JRE, если их нет, отдавая предпочтение JRE.

2) Скачать бинарник**:** http://jmeter.apache.org/download_jmeter.cgi#binaries.

3) Распаковать. Идем в bin. Запустить jmeter.bat.

Готово!

Если что-то идет не так: https://loadtestweb.info/2017/10/17/install-apache-jmeter/

c) Под Linux:

1) Установить JDK или JRE, отдавая предпочтение JRE.

2) Скачать бинарник: http://jmeter.apache.org/download_jmeter.cgi#binaries.

3) Распаковать.

4) Перейти в bin.

5) Запустить jmeter.bat (либо вместо jmeter.bat -> jemeter.sh).

Готово!

Если что-то идет не так: https://loadtestweb.info/2017/10/17/install-apache-jmeter/

Начало работы с JMeter:

a) Создать “Thread Group”.

Thread Group, управляющая такими настройками, как количество потоков, используемых для тестирования и количество запросов в тесте, находится в категории **Treads (Users).**

Кликнув на Test Plan -> Add -> Threads(Users) -> Thread Group. На рисунке 7.3 представлено добавление “Thread Group”.

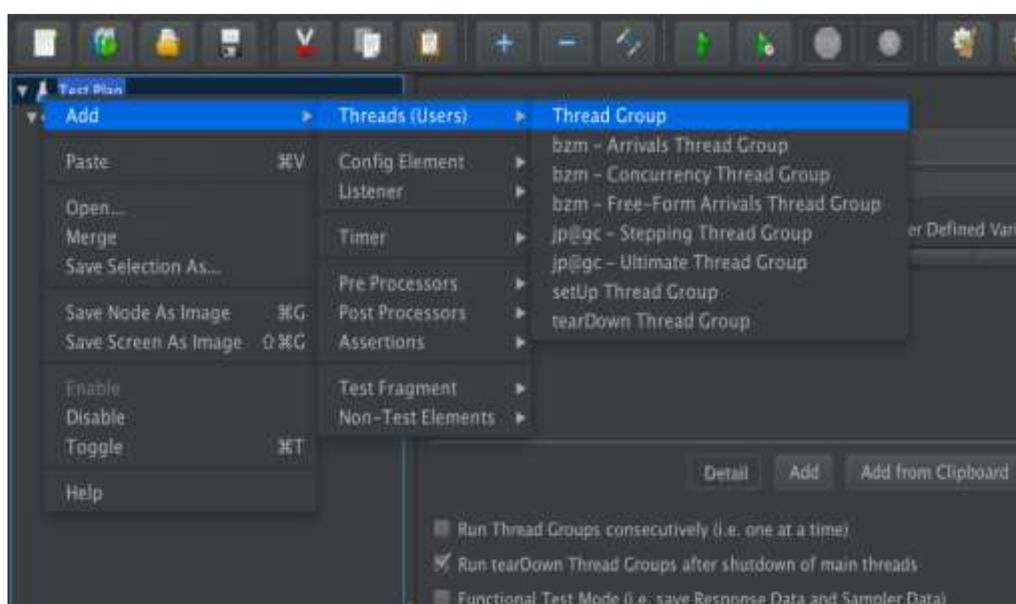


Рисунок 7.3 – Добавление “Thread Group”

b) Настройка Thread Group

Все опции для начала выставляем в единицу. Это один виртуальный пользователь, который один раз выполнит сценарий (в случае используемого нами HTTP Request Defaults'а — выполнит один запрос, соответствующий первой строчке лога). Для отладки теста больше и не понадобится.

Опция «Number of Threads» указывает сколько пользователей будут обращаться к серверу, «Ramp-Up Period» указывает за сколько все эти пользователи будут подключены.

Thread Group предоставляет планировщик (scheduler). Кликнув на чекбокс внизу Thread Group панели, можно активировать/дезактивировать (enable/disable) дополнительные поля, в которых можно указать длительность теста и отсрочку начала. Duration контролирует продолжительность каждой группы потоков (thread group). Startup Delay указывает после какого количества времени начнется процесс. Когда тест запускается, JMeter ждет Startup Delay (seconds) перед началом потоков (Threads) из Thread Group, и запускает на сконфигурированное время (Duration). На рисунке 7.4 представлено окно настройки “Thread Group”.

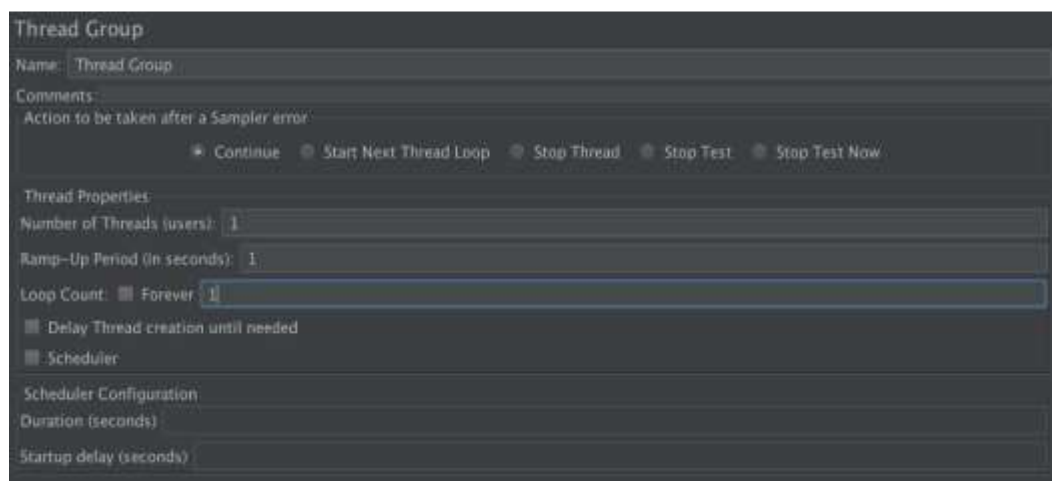


Рисунок 7.4 – Окно настройки “Thread Group”

c) Добавляем и настраиваем HTTP Request Defaults

Левым кликом по «Thread Group» добавляем HTTP Request Defaults (Add -> Config Element -> HTTP Request Defaults). На рисунке 7.5 представлено добавление “HTTP Request Defaults”

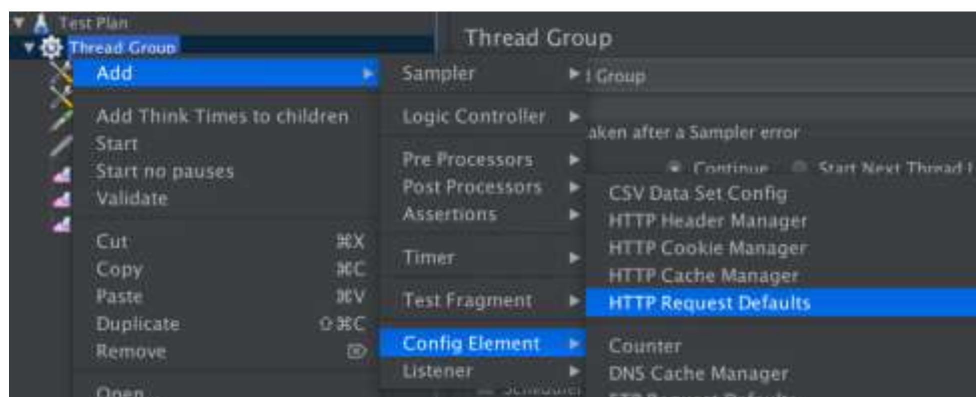


Рисунок 7.5 – Добавление “HTTP Request Defaults”

Для начала необходимо сконфигурировать подключение к серверу (его адрес и порт будет у человека, который запустил у себя сервер). На рисунке 7.6 представлено конфигурирование “HTTP Request Defaults”.

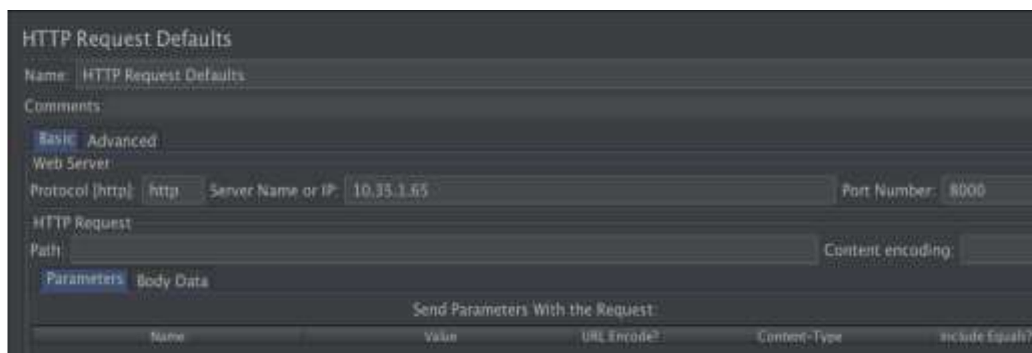


Рисунок 7.6 – Конфигурирование “HTTP Request Defaults”

d) Добавляем и настраиваем HTTP Header Manager

Левым кликом по «Thread Group» добавляем HTTP Header Manager (Add -> Config Element -> HTTP Header Manager). На рисунке 7.7 представлено добавление “ HTTP Header Manager”.

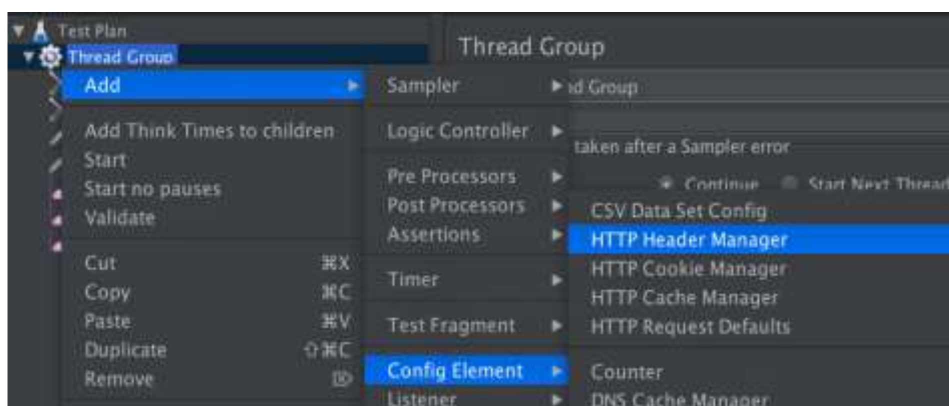


Рисунок 7.7 – Добавление “HTTP Header Manager”

Для POST запросов необходимо добавить Header “Content-Type” со значением “application/json”. Для этого необходимо нажать на кнопку “Add”, и затем двойным кликом начать редактирование каждого из полей в появившейся строке. На рисунке 7.8 представлено конфигурирование “ HTTP Header Manager”.

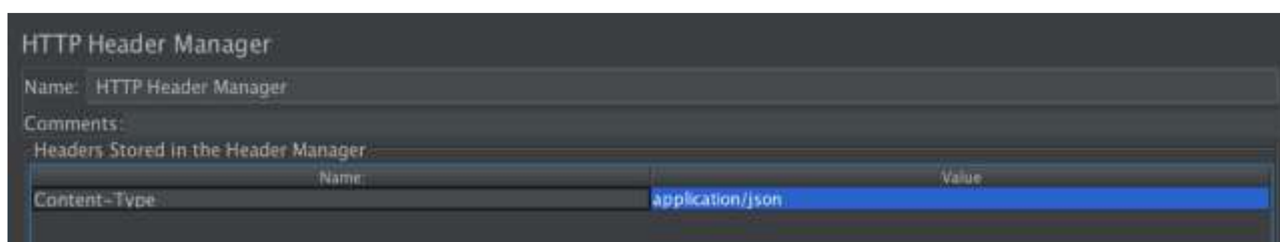


Рисунок 7.8 – Конфигурирование “ HTTP Header Manager”

e) Добавляем и настраиваем View Results in Table

Левым кликом по «Thread Group» добавляем View Results in Table (Add -> Listener-> View Results in Table). Рисунок 7.9 иллюстрирует добавление “View Results in Table”.

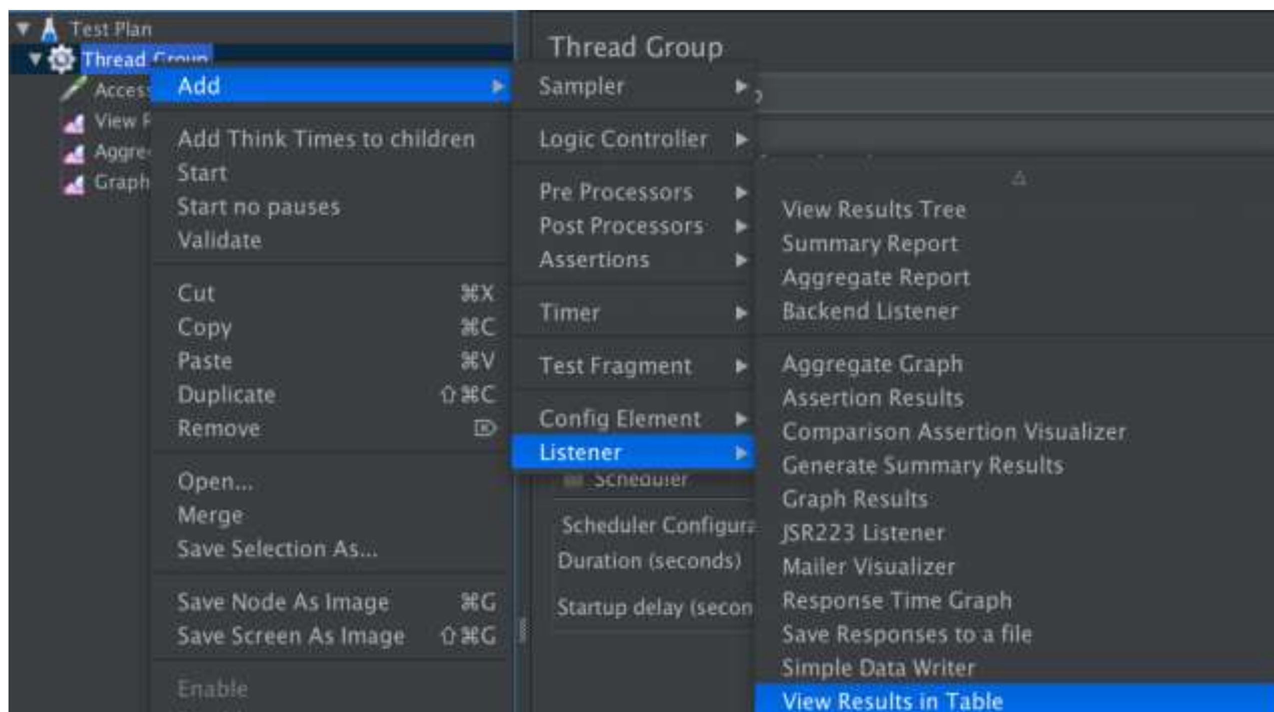


Рисунок 7.9 – Добавление “View Results in Table”

Этот инструмент собирает информацию по запросам и формирует их в виде таблицы. Он позволяет записывать результаты испытаний в log файл. Лучше самим создать его и указать к нему путь, чтобы иметь к нему доступ. По умолчанию он будет создан в папке JMeter-а. Так же можно выбрать какие именно результаты хотим занести и отобразить: Ошибки (Errors) и Успешные прохождения (Successes). Нажав на кнопку “Configure”, можно добавить и убрать определенные столбцы. Интерфейс “View Results in Table” представлен на рисунке 7.10.

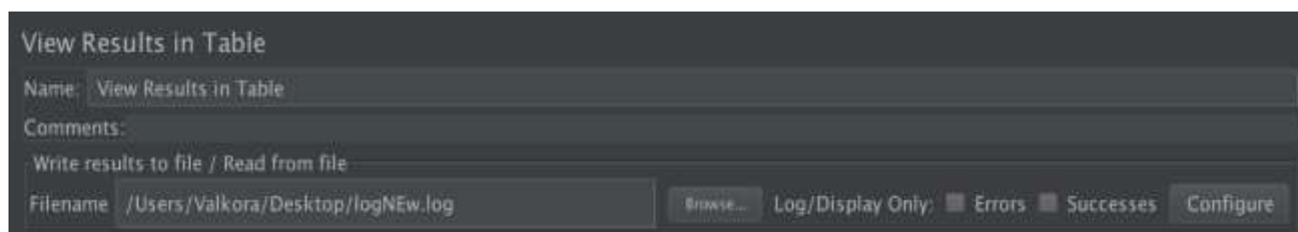


Рисунок 7.10 – Интерфейс “View Results in Table”

После запуска испытаний результат работы“View Results in Table” представлен на рисунке 7.11.

View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename /Users/Valkora/Desktop/logNEw.log Browse... Log/Display Only: Errors Successes Configure

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(...
1	21:36:27.124	Thread Grou...	http://:8000...	31		2177	0	0	0
2	21:36:27.169	Thread Grou...	http://:8000...	4		2177	0	0	0
3	21:36:27.238	Thread Grou...	http://:8000...	1		2177	0	0	0
4	21:36:27.300	Thread Grou...	http://:8000...	1		2177	0	0	0

Рисунок 7.11 – Результат работы “View Results in Table”

f) Добавляем и настраиваем Aggregate Report

Левым кликом по «Thread Group» добавляем HTTP Request Defaults (Add -> Listener -> Aggregate Report). Процесс добавления представлен на рисунке 7.12.

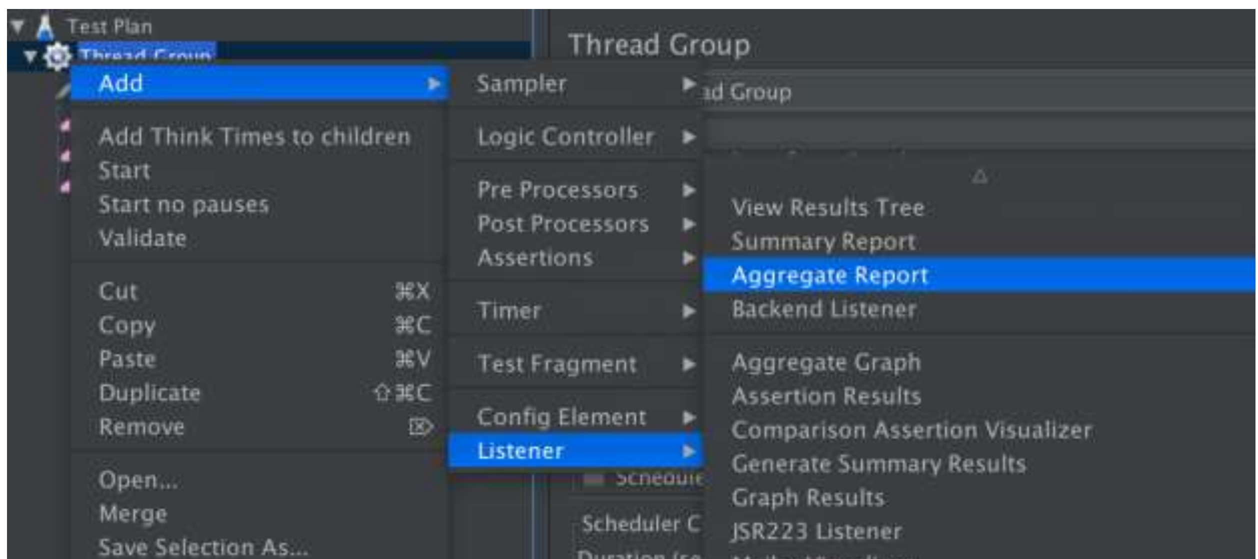


Рисунок 7.12 – Добавление “Aggregate Report”

Этот инструмент собирает сводные показатели по запросам и формирует их в виде таблицы. Сводные показатели проводятся для каждого эндпоинта, а затем есть завершающая строка Total. Этот инструмент так же позволяет записывать результаты испытаний в log файл. Лучше самим создать его и указать к нему путь, чтобы иметь к нему доступ. По умолчанию он будет создан в папке JMeter-а. Интерфейс “Aggregate Report” виден на рисунке 7.13.

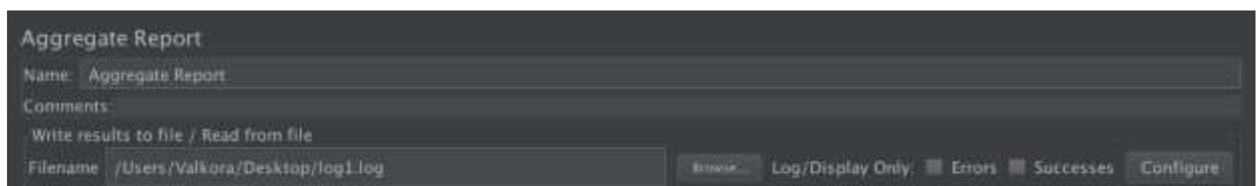


Рисунок 7.13 – Интерфейс “Aggregate Report”

Среди колонок для данной таблицы будет средняя пропускная способность для каждого эндпоинта, максимальное и минимальное время задержки в миллисекундах и другие. Результат “Aggregate Report” представлен на рисунке 7.14.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Through...	Received ...	Sent KB/...
http://10.35.1.65:8000/notes	1000	18071	20399	37029	40757	47037	119	108310	49.40%	8.6/sec	14.35	0.00
TOTAL	1000	18071	20399	37029	40757	47037	119	108310	49.40%	8.6/sec	14.35	0.00

Рисунок 7.14 – Результат “Aggregate Report”

g) Добавление и настройка “Graph Result”

Левым кликом по «Thread Group» добавляем Graph Result (Add -> Listener -> Graph Result). Процесс добавления представлен на рисунке 7.15.

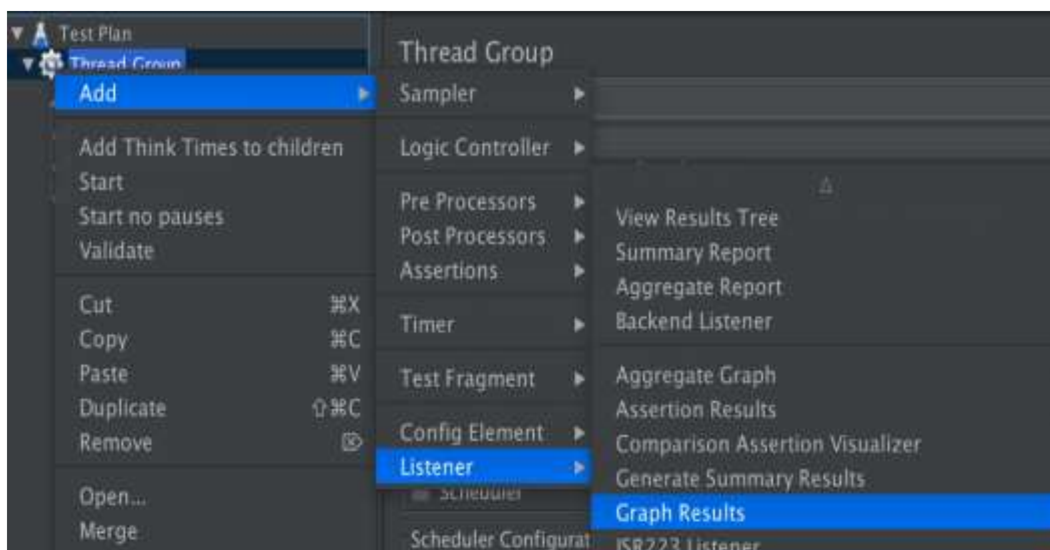


Рисунок 7.15 – Добавление “Graph Result”

Этот инструмент помогает визуализировать происходящий процесс. Так же он позволяет записывать результаты испытаний в файл. Лучше самим создать его и указать к нему путь, чтобы иметь к нему доступ. По умолчанию он будет создан в папке JMeter-а. На рисунке 7.16 можно увидеть интерфейс “ Graph Result”.

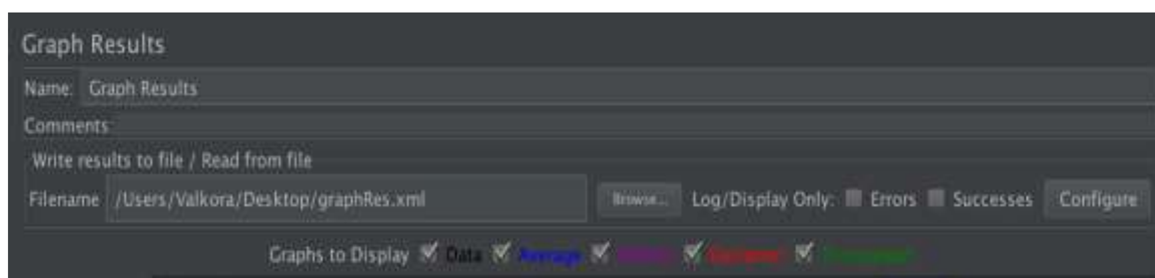


Рисунок 7.16 – Интерфейс “ Graph Result”

На рисунке 7.17 представлен результат работы “Graph Result”. Ниже графика представлены значения параметров. Среди них можно найти интересующую нас пропускную способность (Throughput).

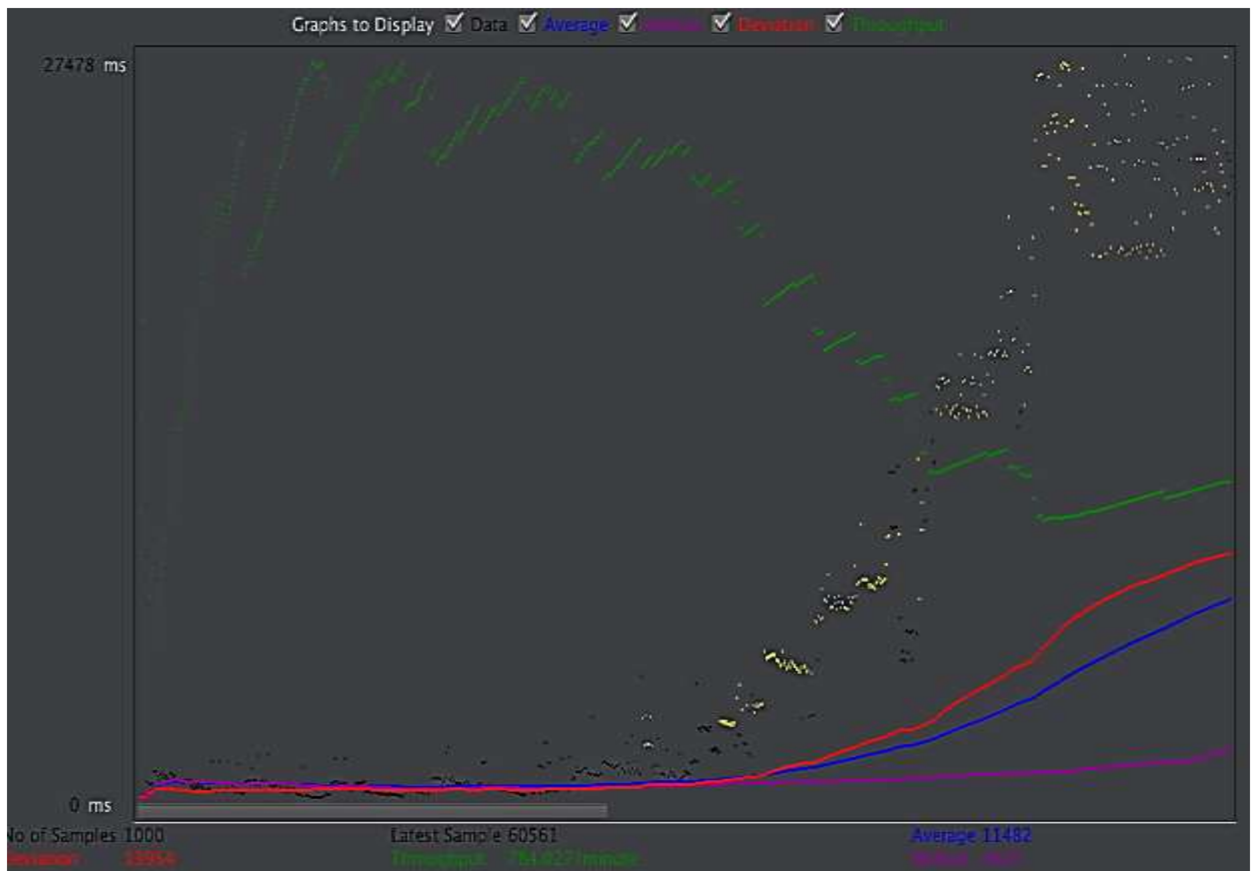


Рисунок 7.17 – Результат “Graph Result”

h) Добавление HTTP Request.

Левым кликом по «Thread Group» добавляем Graph Result (Add -> Sample -> HTTP Request). Процесс добавления представлен на рисунке 7.18.

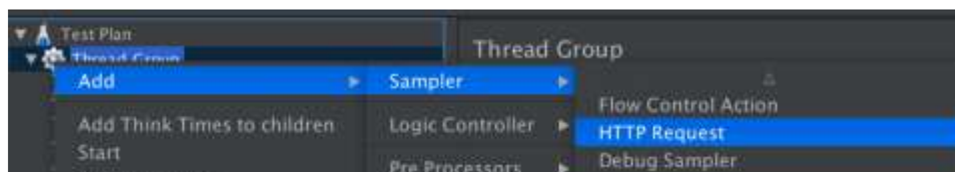


Рисунок 7.18 – Добавление “HTTP Request”

Теперь необходимо выбрать метод, первым добавим GET запрос и указать путь(path) “/notes”. На рисунке 7.19 можно увидеть конфигурацию “HTTP Request”.

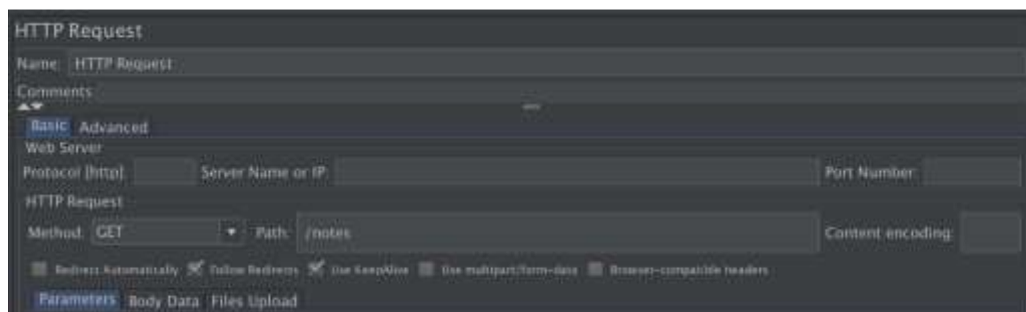


Рисунок 7.19 – Конфигурация “HTTP Request” для “GET” метода

Теперь добавим POST запрос и укажем путь (path), в данном случае он такой же, как и у GET - “/notes”. Затем необходимо добавить тело запроса. Для этого нужно перейти на вкладку “Body Data” и добавить JSON с ключами “title” и “body”. На рисунке 7.20 представлена конфигурация “HTTP Request” для POST запроса.

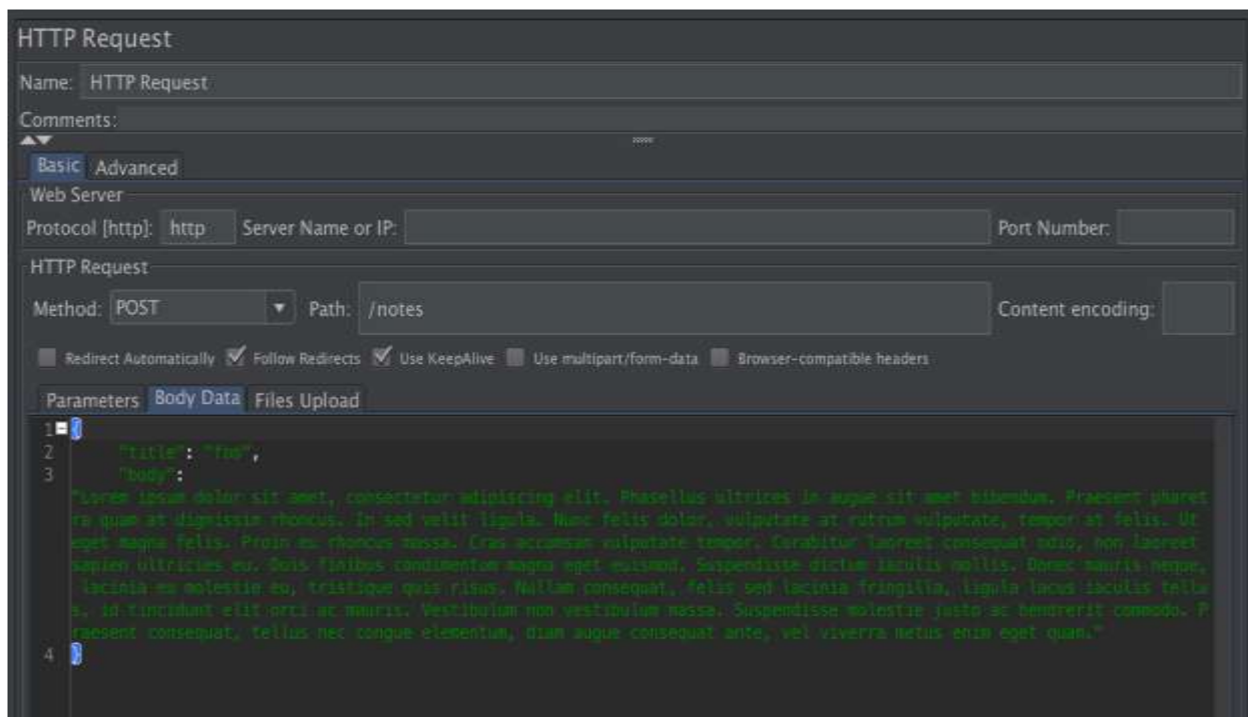


Рисунок 7.20 – Конфигурация “HTTP Request” для “GET” метода

По образцу можно будет добавить и остальные endpoint-ы. Их список можно будет найти в Приложении.

i) Панель инструментов

В панели инструментов нас в основном будут интересовать, представленные на рисунке 7.21, возможности. Запустить все можно с помощью зеленной кнопки “Play”. Во время выполнения тестов их можно остановить, нажав на “Stop” или “Shutdown”, которые становятся красными в активном режиме. С помощью “Clear” и “Clear All” можно очистить конкретный элемент или очистить всё соответственно.

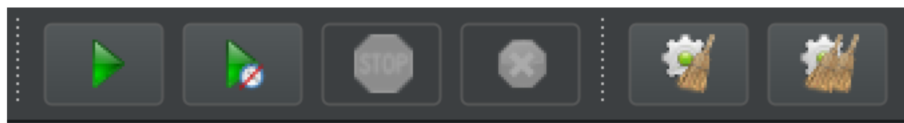


Рисунок 7.21 – Часть Панели инструментов

Так же, при желании, можно запустить не все endpoint-ы, а только некоторые. Убрать endpoint можно правым кликом и выбрать опцию “Disable”. Процесс представлен на рисунке 7.22.

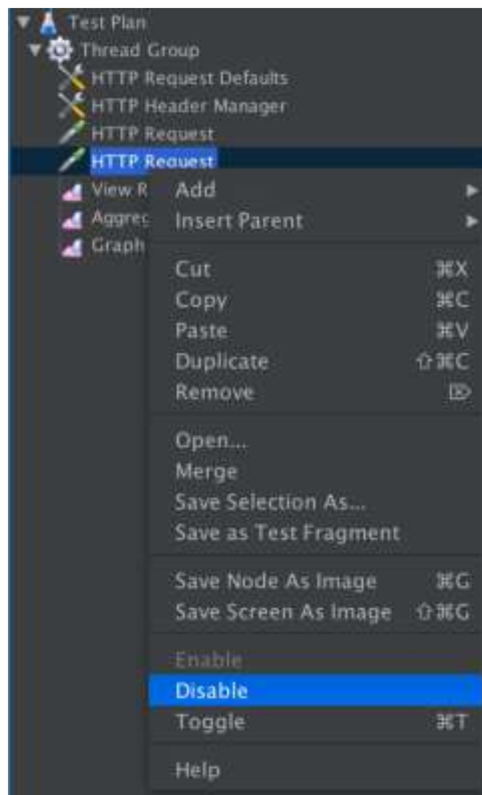


Рисунок 7.22 – Операции над файлами

ЛАБОРАТОРНАЯ РАБОТА № 8

Тема: Изучение систем отслеживания ошибок.

Цель работы: освоение трёх видов систем отслеживания ошибок: MS TFS, Atlassian Jira и Zenhub. Составление отчета о дефекте.

Система отслеживания ошибок (англ. bug tracking system) — прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, тестировщикам и др.) учитывать и контролировать ошибки (баги), найденные в программах, а также следить за процессом устранения этих ошибок.

В настоящее время существует огромное количество систем отслеживания ошибок, выбор которых зависит от специфики, размеров проекта, а также от личных предпочтений команды разработки.

Наиболее популярными решениями являются системы Atlassian Jira, Team Foundation Server от Microsoft, а также более простые системы, например, ZenHub от GitHub.

Каждая система из вышеперечисленных обладает своими особенностями. Можно отметить, что TFS от Microsoft единственная требует установки программного обеспечения на локальный компьютер. Это может послужить недостатком при выборе системы отслеживания ошибок. Однако стоит отметить, что данная система поддерживает проведение ручного тестирования, что сильно отличает ее от других систем.

Jira от Atlassian обладает большим функционалом, однако ее подписка является платной. Поэтому данную систему используют в основном на больших корпоративных проектах. В свою очередь хорошим вариантом для небольших команд является встроенный плагин для GitHub, который называется ZenHub. Эта система включает в себя достаточный функционал для своей доступности. Она является бесплатной и очень просто подключается к репозиторию GitHub.

Team Foundation Server

Team Foundation Server (TFS) — комплексное решение от Microsoft, которое включает в себя систему управления версиями, сбор данных, построение отчетов, отслеживание статусов и изменений по проекту.

Данная система является одной из самых богатых систем багтрекинга по функционалу. Также огромным плюсом является возможность ее интеграции с процессом тестирования на проекте. TFS в связке с ПО Test Manager от Microsoft представляет собой очень удобную систему для проведения тестирования.

Microsoft Test Manager (MTM) — часть этого продукта и требует установки Visual Studio. Становится возможным связать задачи, которые поставлены перед исполнителем, с заведенными репортами (дефектами) и отчетами о затраченном на работу времени.

Планы и результаты тестирования сохраняются на сервере Team Foundation Server. MTM включает в себя тест-план, тест-кейс и конфигурации.

Сам TFS является проприетарным ПО, лицензия — коммерческая. Работает на трех уровнях: клиентский уровень, прикладной уровень и уровень данных, в зависимости от чего возможна работа либо через web, либо только через десктоп-приложение. MTM работает только на прикладном уровне, поэтому требуется установка на сервер (если сервер удаленный, работа проводится через VPN).

Установка и базовая конфигурация Microsoft Team Foundation Server

Для запуска установки TFS необходимо скачать установочный файл с официального сайта Microsoft (https://www.microsoft.com/en-us/download/details.aspx?id=48260). После успешной загрузки открываем установщик и следуем инструкциям мастера установки, пример которого представлен на рисунке 8.1.

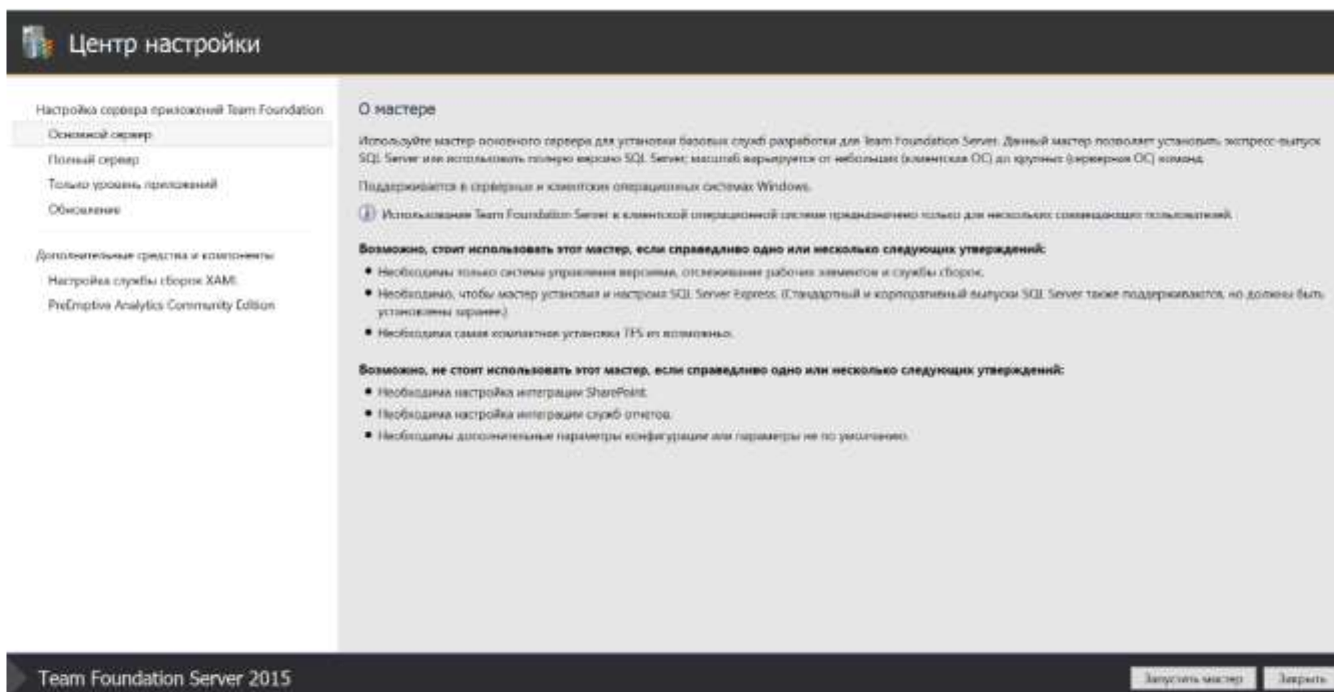


Рисунок 8.1 – Запуск мастера настройки TFS

Далее следуем шагам мастера и выбираем нужные опции. Так, необходимо разрешить мастеру установить SQL Server Express и использовать системную учетную запись для сборки и входа в систему. Настройки этого шага и параметры конфигурации представлены на рисунках 8.2 и 8.3.

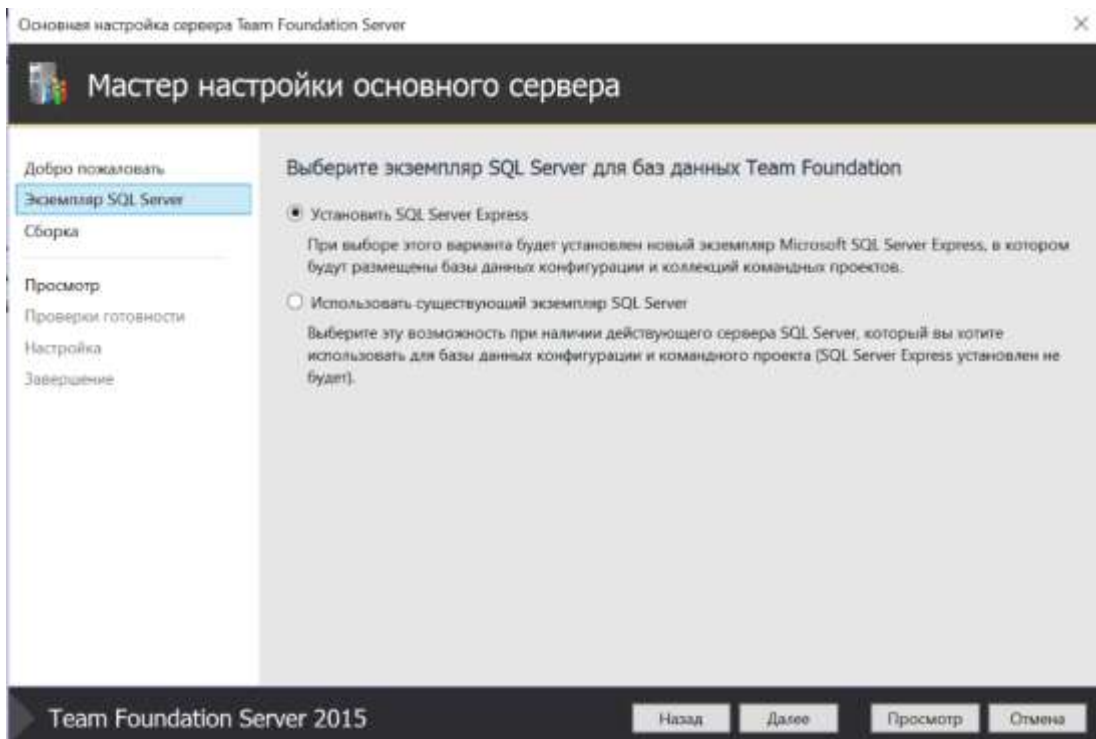


Рисунок 8.2 – Мастер настройки сервера MS TFS

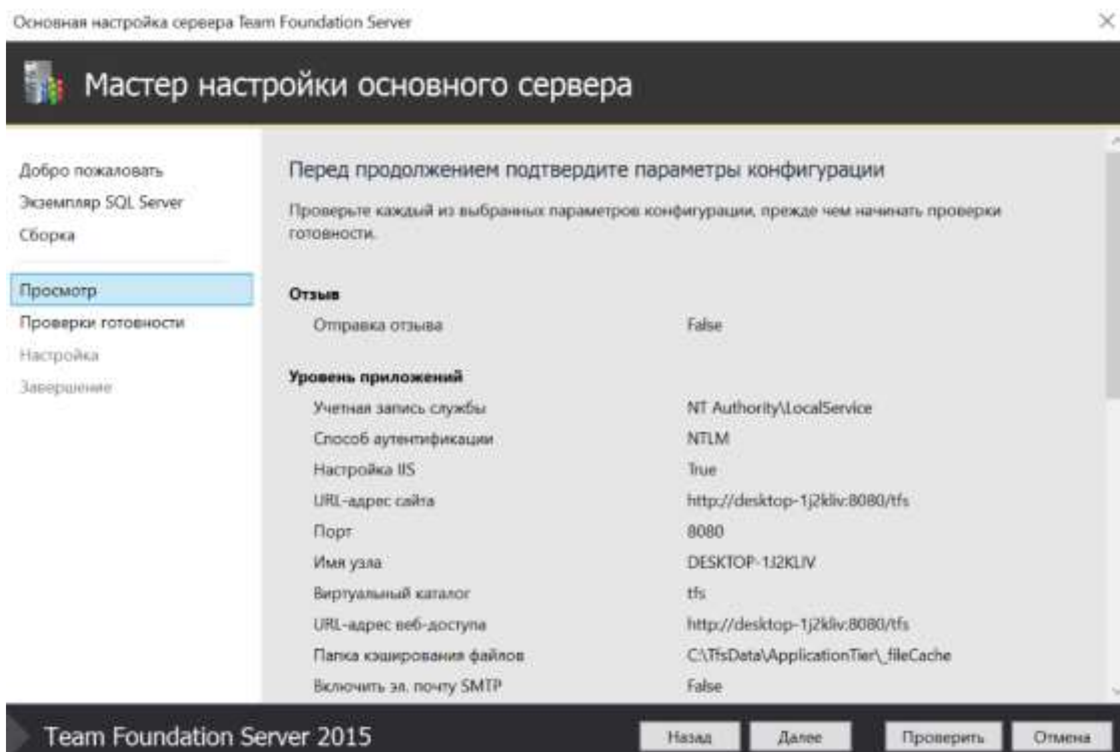


Рисунок 8.3 – Параметры конфигурации сервера TFS

В случае, если при установке или конфигурации сервера будет необходимо ввести данные учетной записи, необходимо указать данные учетной записи пользователя системы.

После успешного завершения работы мастера конфигурации сервер будет доступен по следующей ссылке: localhost:8080/tfs/. Стартовая страница запуска MS TFS представлена на рисунке 8.4.

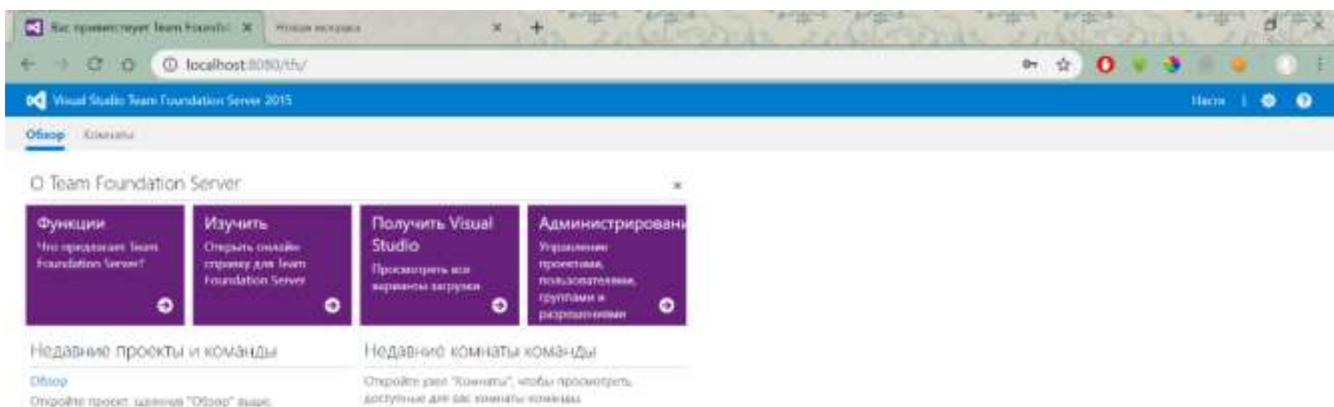


Рисунок 8.4 – Стартовая страница MS TFS

Чтобы начать работу с системой TFS, необходимо создать новый проект, где будут храниться все задачи, спецификации и дефекты. Для этого зайдем в Microsoft Visual Studio и создадим новое подключение в Team Explorer. В открытом окне, представленном на рисунке 8.5, нужно ввести URL ранее установленного сервера MS TFS.

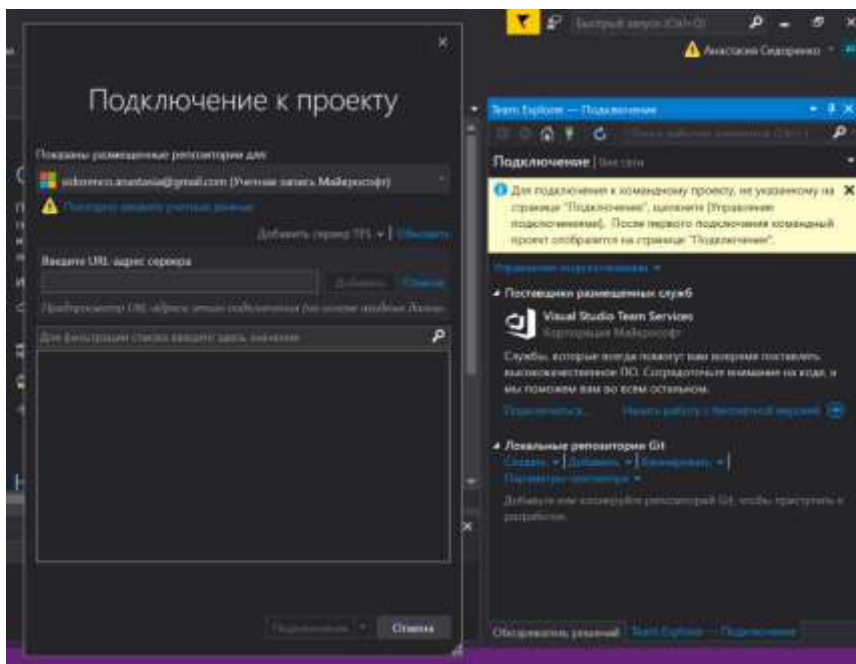


Рисунок 8.5 – Создание нового подключения

После того, как подключение успешно создано, можно создать новый проект. Для этого в свойствах сервера выберем пункт Проект и мои команды – Создать командный проект… Более подробно этот шаг показан на рисунке 8.6.

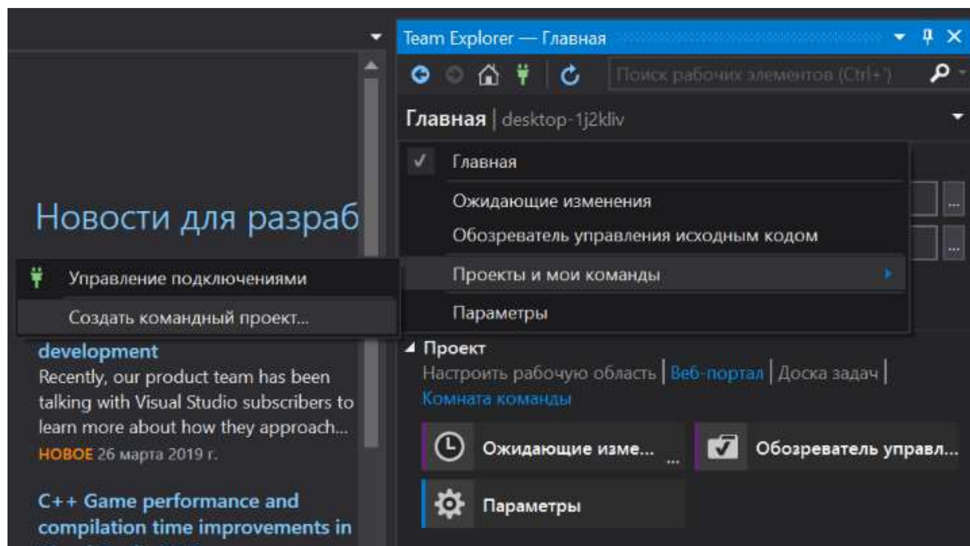


Рисунок 8.6 – Создание нового проекта

В открытом окне введем название нового командного проекта и выберем шаблон процессов из трех представленных: Scrum, Agile, CMMI. Подробнее про отличия этих типов можно прочитать в документации MS TFS (https://docs.microsoft.com/en-us/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops). Основными отличиями между этими шаблонами являются различие в процессах и жизненном цикле задач на доске, а также в основных типах задач в проекте. Для выполнения данной лабораторной работы не важно, какой именно шаблон был выбран, поэтому для примера можно установить шаблон процессов Scrum. На рисунке 8.7 представлен пример параметров нового проекта.

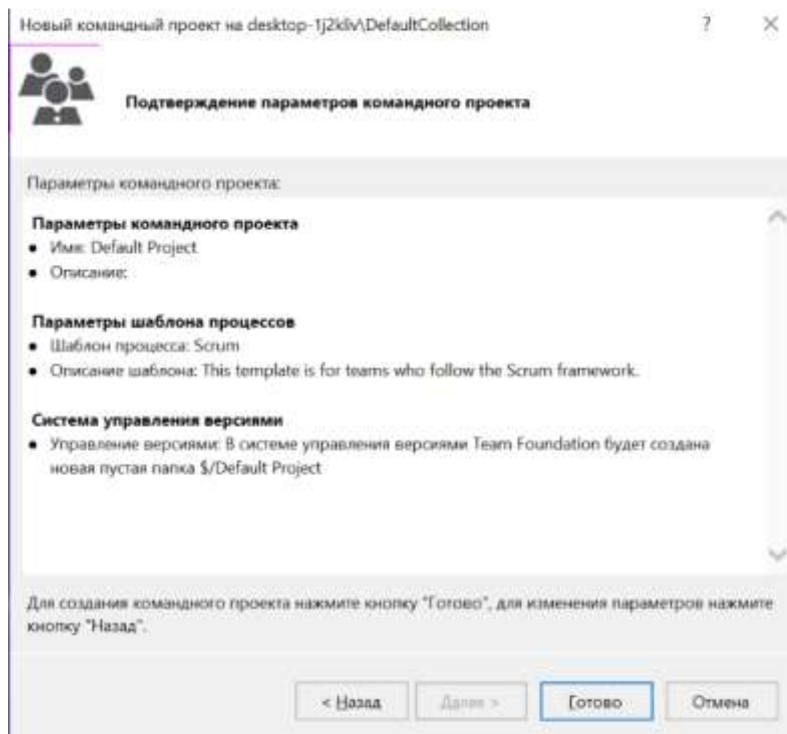


Рисунок 8.7 – Параметры нового проекта

Теперь созданный проект будет доступен в веб версии MS TFS. Для того, чтобы зайти в детали проекта, необходимо нажать на кнопку Обзор стартовой страницы TFS. После этого будет доступен необходимый функционал для дальнейшей работы. Стартовая страница проекта MS TFS представлена на рисунке 8.8.

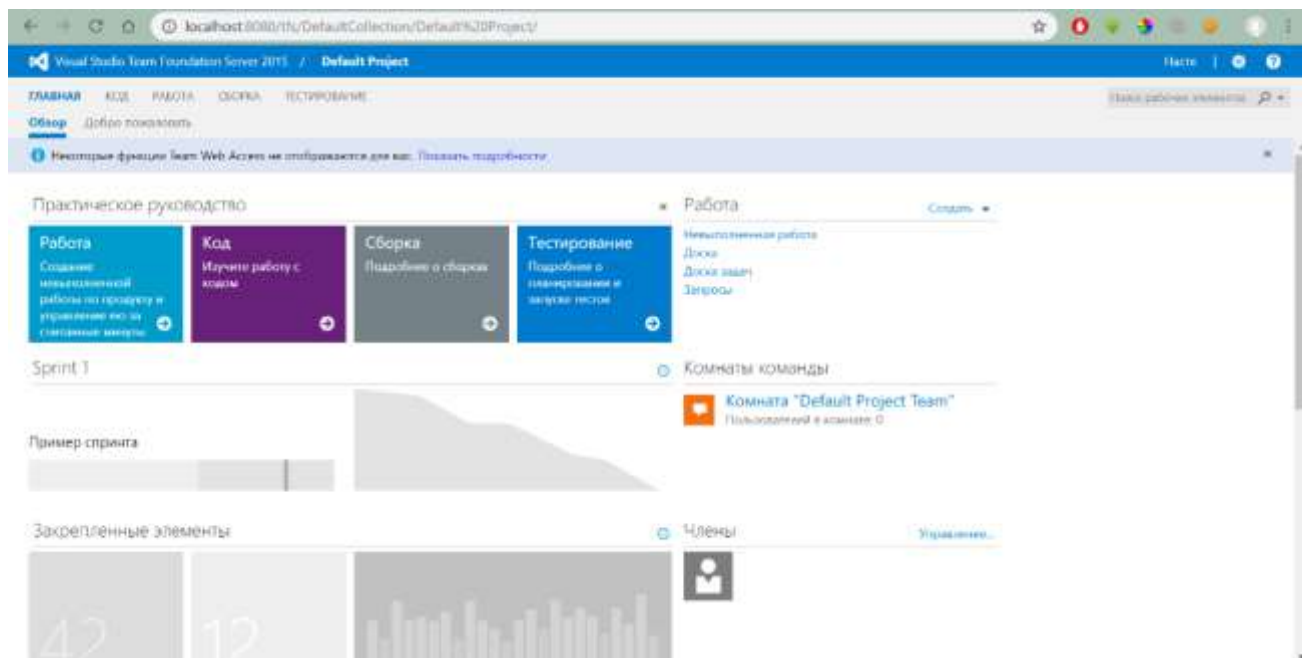


Рисунок 8.8 – Стартовая страница проекта в MS TFS

Можно заметить, что для проекта доступны несколько различных вкладок. В данной лабораторной работы будем использовать вкладки Работа и Тестирование. На вкладке Работа находятся все текущие рабочие элементы: задачи, спецификации и дефекты. Отличительной особенностью современных систем отслеживания ошибок является наличие, так называемой, доски, где очень наглядно представлены все текущие элементы проекта.

Для того, чтобы добавить новый элемент на доску, необходимо зайти на вкладку Работа – Доска и нажать на кнопку Создать новый элемент. В выпадающем меню находятся два типа элементов: Product Backlog Item (фрагмент спецификации, к которому могут быть привязаны задачи для выполнения, а также тестовые варианты) и Bug (дефект).

Создадим новый Product Backlog Item для тестирования веб-сайта airport.md, форма которого показана на рисунке 8.9.

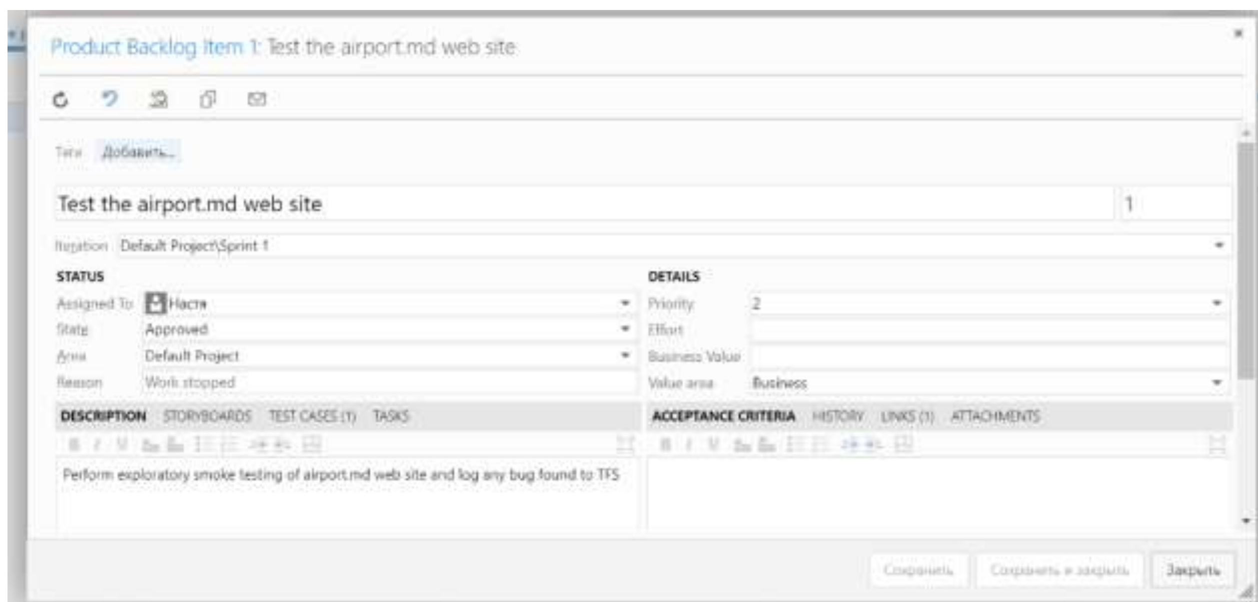


Рисунок 8.9 – Новый рабочий элемент

В случае нахождения дефекта необходимо создать новый отчет о дефекте: создать новый элемент – Bug, который заполняется соответствующим образом. На рисунке 8.10 представлена заполненная форма отчета о дефекте в системе MS TFS. Далее представлен пример отчета о дефекте. Стоит отметить, что наличие скриншотов и других деталей воспроизведения дефекта только приветствуется.

Пример

Название: Система принимает неправильное имя пользователя при регистрации

Описание дефекта: согласно требованиям, имя пользователя должно состоять из цифр и букв. Однако, система принимает имя пользователя, не соответствующее заявленным требованиям, которое состоит только из буквенных символов.

Платформа: Windows 10, Internet Explorer.

Шаги воспроизведения:

Шаг 1. Откройте веб-сайт.

Шаг 2. Введите в поле «Имя пользователя» только буквенные символы.

Шаг 3. Введите корректное значение в поле «Пароль».

Шаг 4. Нажмите на кнопку [Зарегистрироваться].

Ожидаемый результат: появляется сообщение об ошибке, и пользователь не зарегистрирован в системе.

Фактический результат: Пользователь успешно зарегистрирован в системе.

Степень важности: Средняя.

Степень серьезности: Высокая.

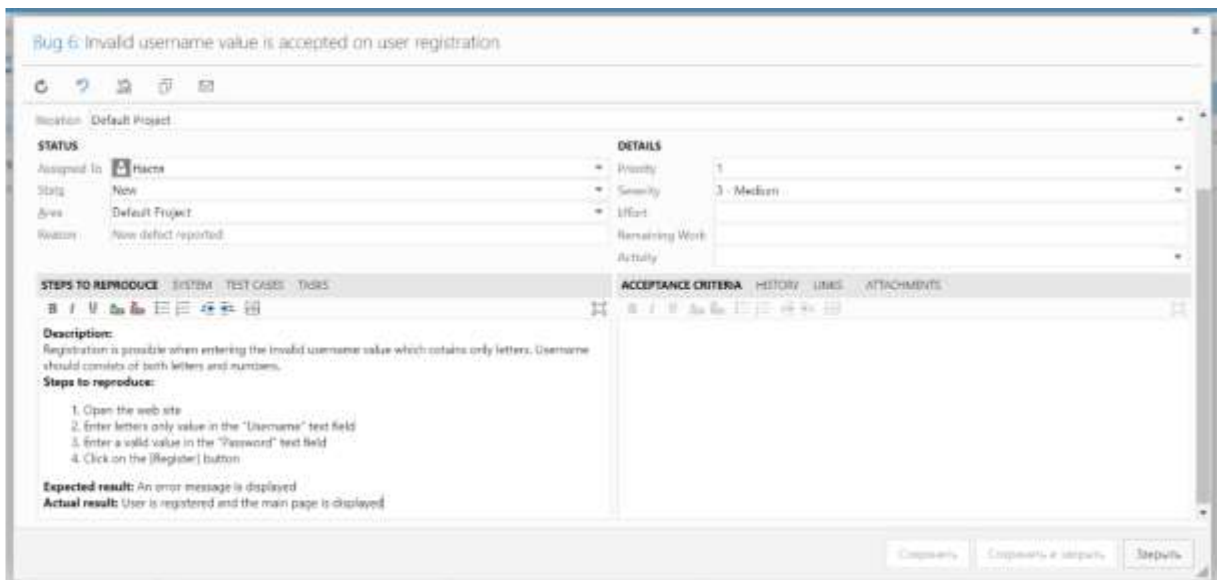


Рисунок 8.10 – Пример отчета о дефекте в системе TFS

ZenHub

ZenHub – расширение для сервиса GitHub (https://www.zenhub.com/), позволяющее собрать несколько репозиториев в один потоковый проект с отчётностью дефектов, проблем, багов. ZenHub имеет как отдельное веб-приложение, так и расширение для браузера Chrome. Чтобы пользоваться сервисом, необходимо иметь профиль на GitHub.

По открытии сайта необходимо нажать “Create New Workspace”, чтобы создать своё рабочее место и затем добавить в него те репозитории, которые вы хотите отслеживать в своём проекте. Окно создания нового рабочего место представлено на рисунке 8.11.

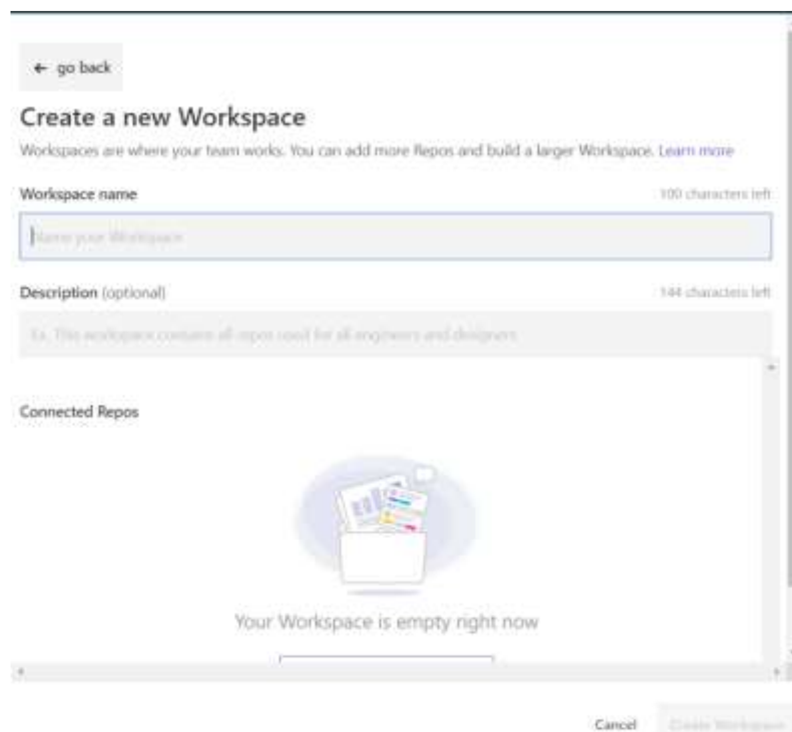


Рисунок 8.11 – Создание рабочего места ZenHub

Отмеченные репозитории отобразятся следующим образом, указанным ниже. Рабочее место ZenHub представлен в виде колонок, работающих в системе drag-and-drop. Доска в системе ZenHub представлена на рисунке 8.12.

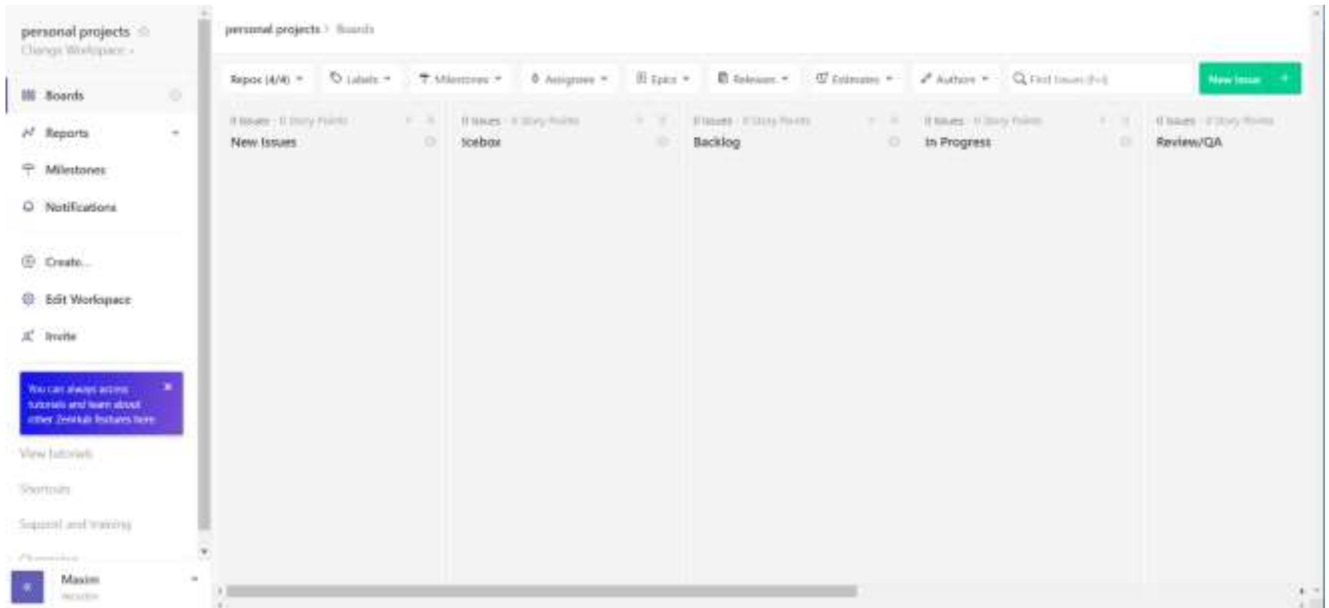


Рисунок 8.12 – Новая рабочая доска ZenHub

Чтобы занести новую проблему, необходимо нажать “New Issue”. Проблеме необходимо дать имя, описание и снимки экрана (по желанию). Можно сразу привязать ответственное лицо (Assignees), крайние сроки исправления (Estimate) и прочее. Пример занесения проблемы показан на рисунке 8.13.

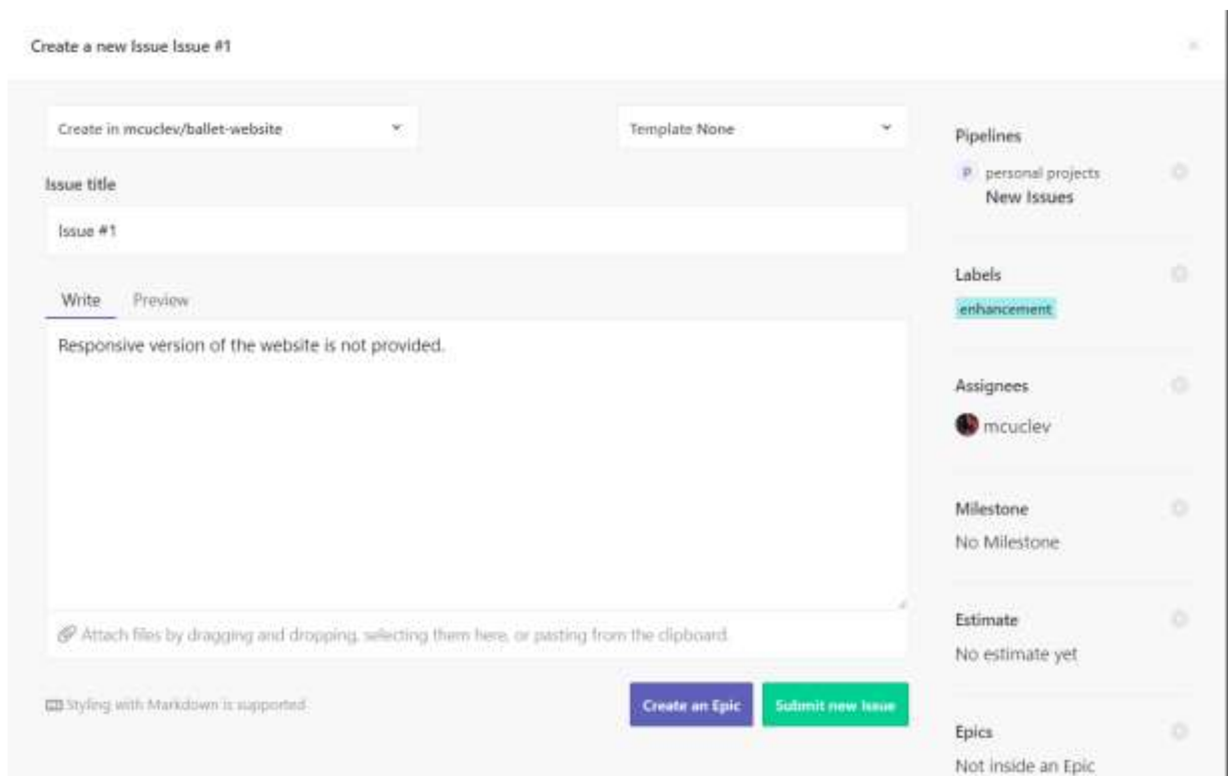


Рисунок 8.13 – Занесение проблемы

После занесения проблемы можно перетаскивать с одной стадии на другие, в зависимости от стадии выполнения, как показано на рисунке 8.14.

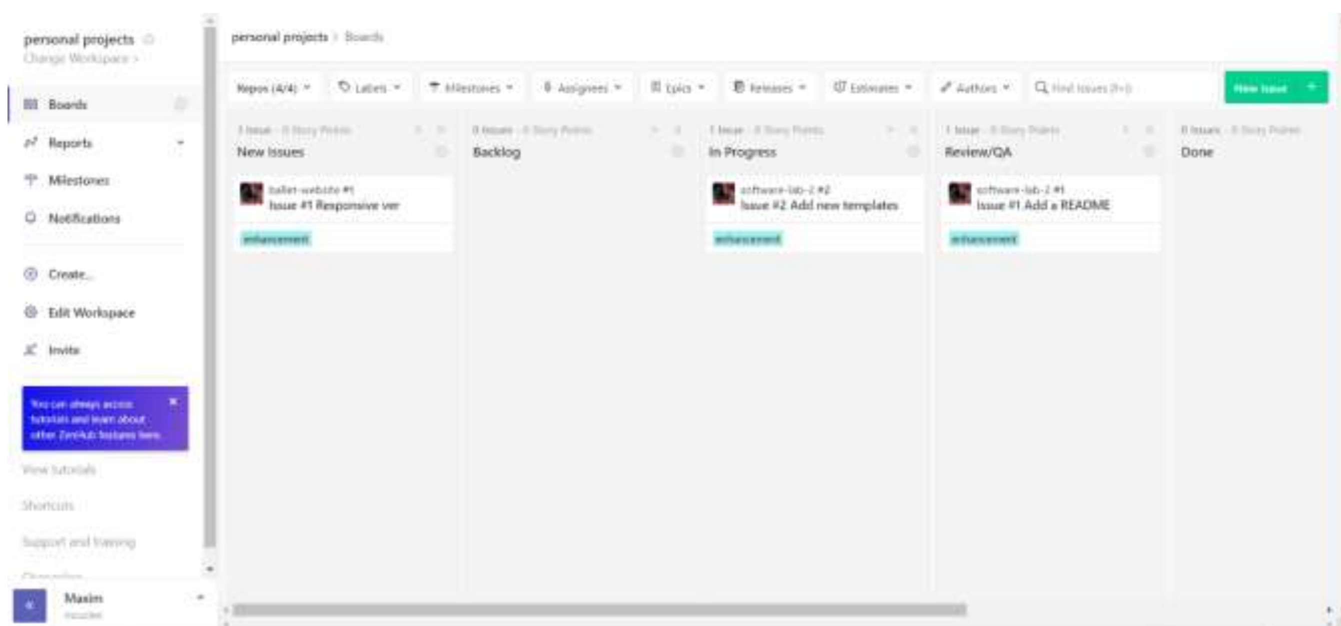


Рисунок 8.14 – Рабочая доска

По завершении, задания можно переносить в секцию Done. В случае, когда решению проблемы препятствуют другие дольше решаемые дефекты, их можно заносить в Icebox или Backlog.

Все проблемы, занесённые в ZenHub, также заносятся и в GitHub, в категории Issues каждого соответствующего репозитория. Список проблем, доступных в GitHub, представлен на рисунке 8.15.

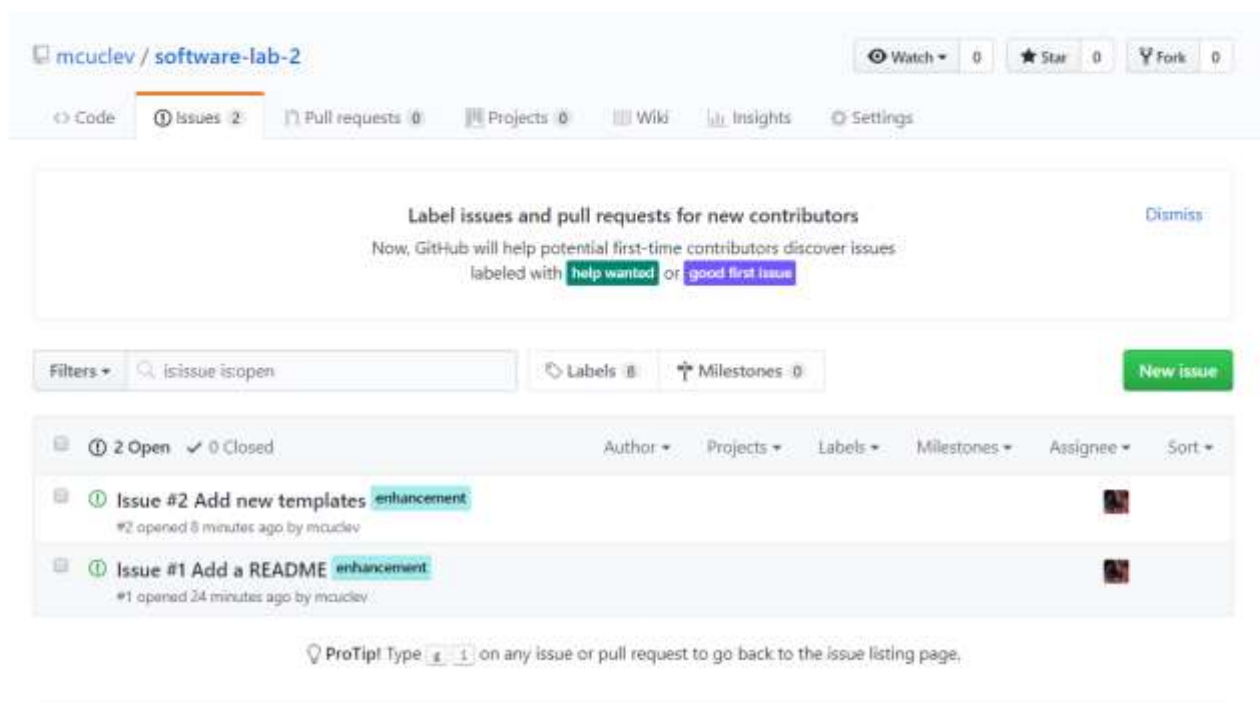


Рисунок 8.15 – Проблемы в GitHub

Atlassian Jira

Atlassian Jira — коммерческая система отслеживания ошибок, предназначена для организации взаимодействия с пользователями, хотя в некоторых случаях используется и для управления проектами.

Основной элемент учёта в системе — задача (англ. ticket или issue). Задача содержит название проекта, тему, тип, приоритет, компоненты и содержание. Задача может быть расширена дополнительными полями (также и новые пользовательские поля могут быть определены), приложениями (например, фотографиями, скриншотами) или комментариями. Задача может редактироваться или просто изменять статус, например, из «открыт» в «закрыт». Какие переходы между состояниями возможны, определяется через настраиваемый поток операций. Любые изменения в задаче протоколируются в журнал.

Регистрация и базовая конфигурация Jira

Jira функционирует на платной основе, но предоставляет и недельный пробный период, на протяжении которого можно ознакомиться и воспользоваться всеми функциями, предоставляемыми данной системой. Для регистрации на пробный период воспользуйтесь следующей ссылкой: https://www.atlassian.com/try/cloud/signup?bundle=jira-software. Страница регистрации в системе Jira представлена на рисунке 8.16.

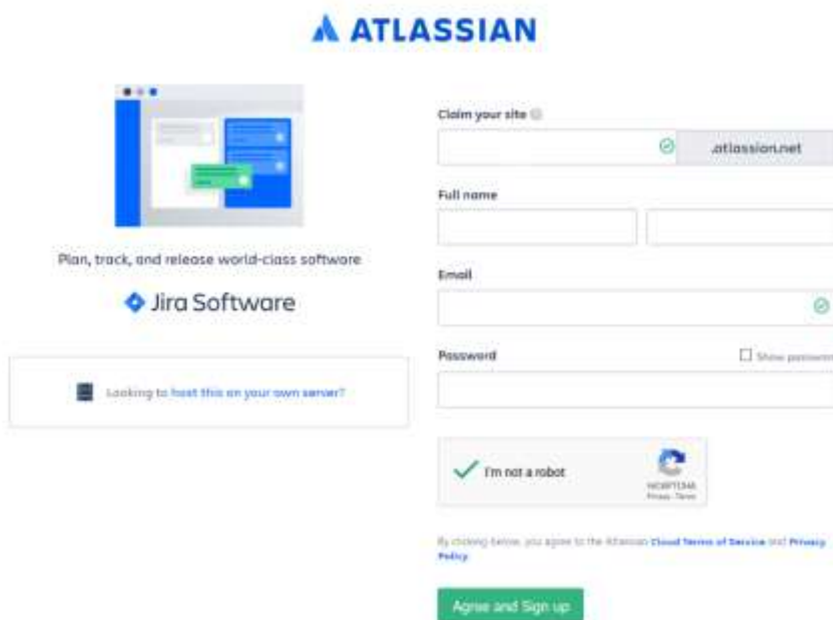


Рисунок 8.16 – Регистрация в Jira

После регистрации необходимо сконфигурировать систему и выбрать наиболее удобную доску. Доска – это страница, на которой отображаются задачи одного или множества проектов, которая позволяет с удобством наблюдать, управлять и создавать отчёты о проделанной работе. В Jira существует несколько типов досок:

a) Next-gen board подходит для команд разработчиков, которые не имеют большого опыта работы с agile. Данная доска наиболее проста в понимании.

b) Scrum board подходит для команд, которые предпочитают работу в спринтах, установленных периодах времени, за которые нужно выполнить определённые задачи. Включает в себя и back log.

c) Kanban board подходит для команд, которые концентрируются на поддержании и управлении текущими задачами. Так же предусматривает back log.

На рисунках 8.17 и 8.18 представлена базовая конфигурация Jira и страница выбора доски.

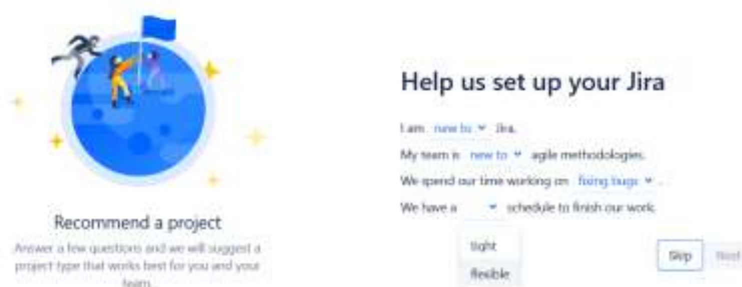


Рисунок 8.17 – Конфигурация Jira

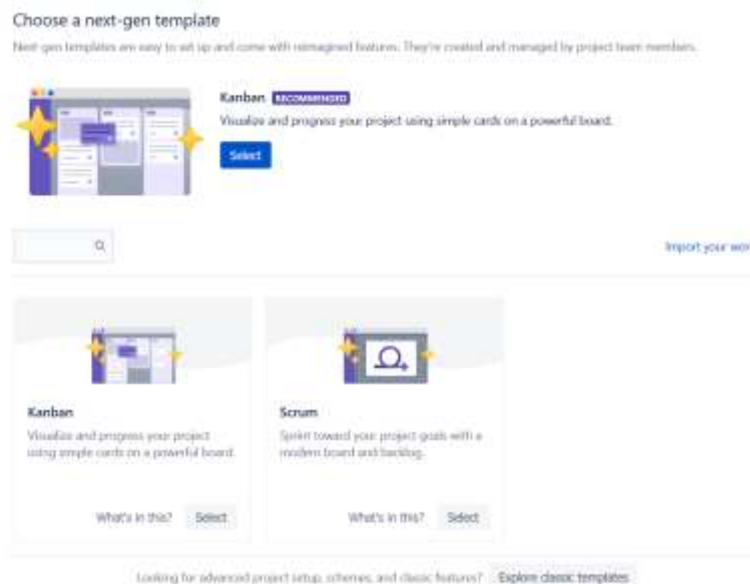


Рисунок 8.18 – Выбор доски

Работа с Jira

При первом входе в систему начальная страница выглядит следующим образом, как показано на рисунке 8.19.

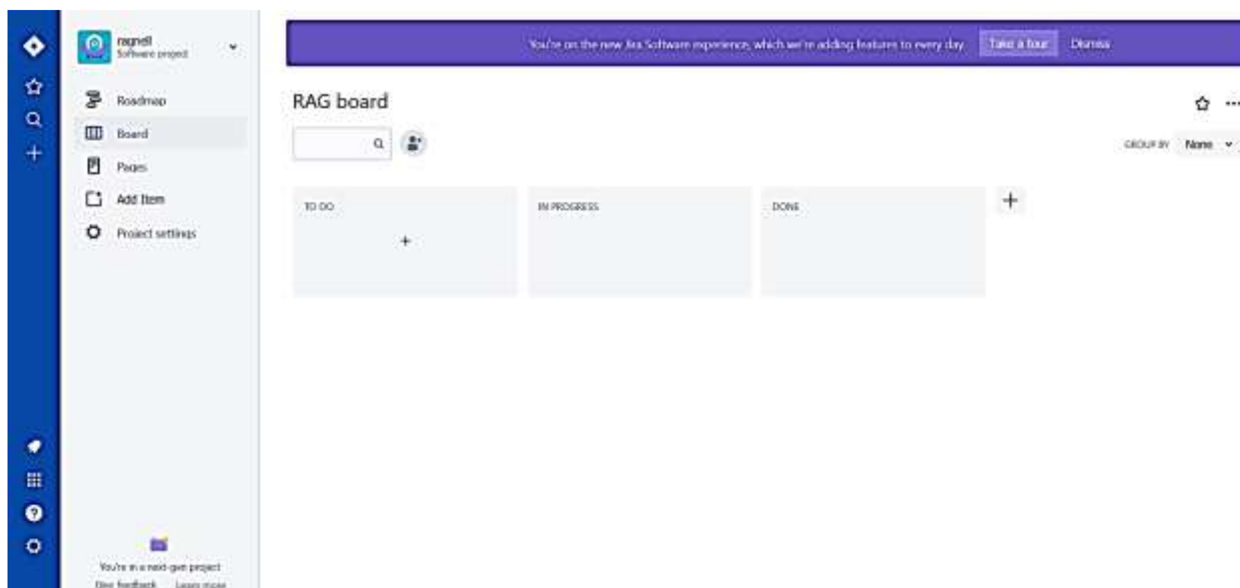


Рисунок 8.19 – Добро пожаловать в Jira!

Работать над каким бы то ни было проектом удобнее в команде. Для того, чтобы создать команду, необходимо пригласить в свою систему пользователей. Делается это в разделе Users. На рисунке 8.20 представлена форма для приглашения нового участника команды.

Рисунок 8.20 – Отправка приглашения в команду

После того, как ваша команда пополнилась новыми участниками, необходимо создать и распределить роли в проекте. Процесс создания новой роли и привязки ее к участнику проекта представлен на рисунках 8.21 и 8.22.

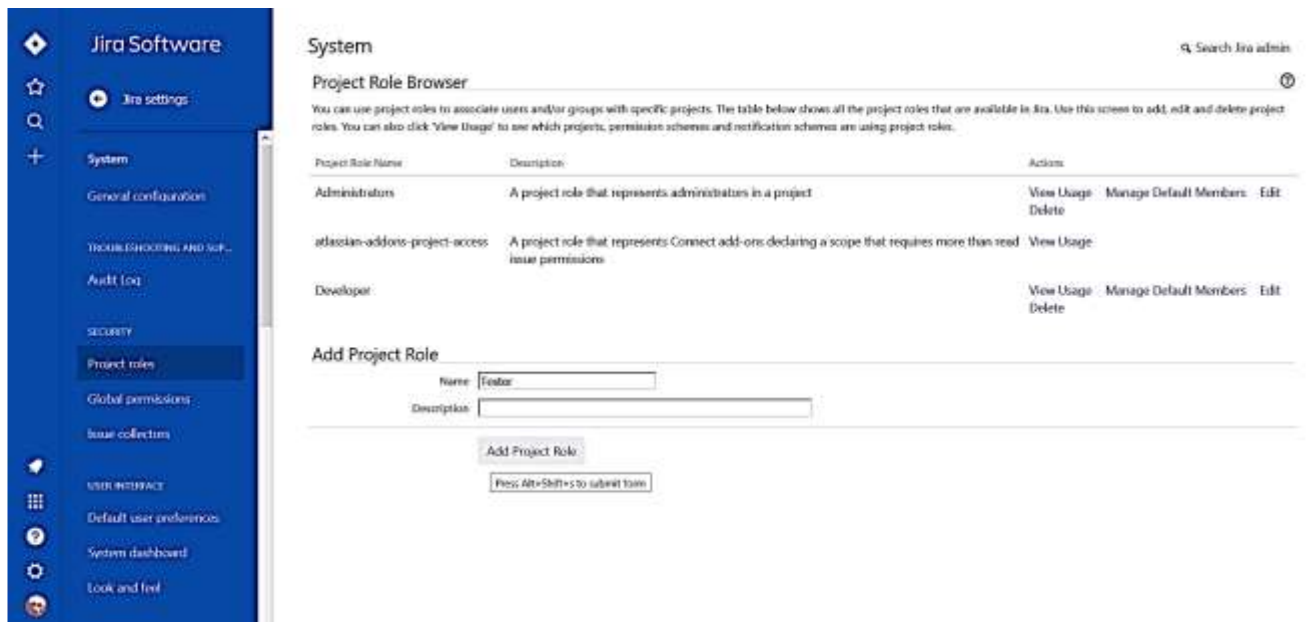


Рисунок 8.21 – Создание ролей

Рисунок 8.22 – Добавление роли участнику проекта

Ещё одно преимущество Jira состоит в возможности настраивать workflow в соответствии с нуждами команды и требованиями к проекту. Workflow включает в себя список состояний, через которые проходят задачи от момента их создания до момента их закрытия. В Jira можно воспользоваться как заданным по умолчанию workflow, так и создать свой собственный.

На рисунках 8.23 и 8.24 представлен пример создания своего собственного workflow и добавления ранее созданного workflow к проекту.

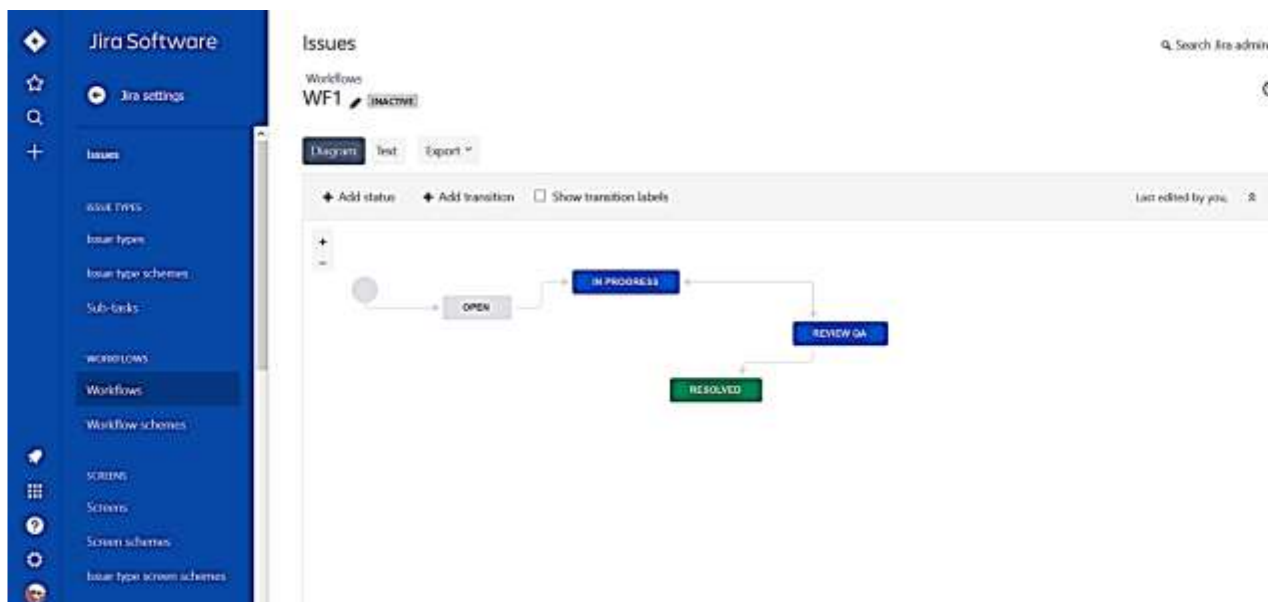


Рисунок 8.23 – Создание собственного Workflow

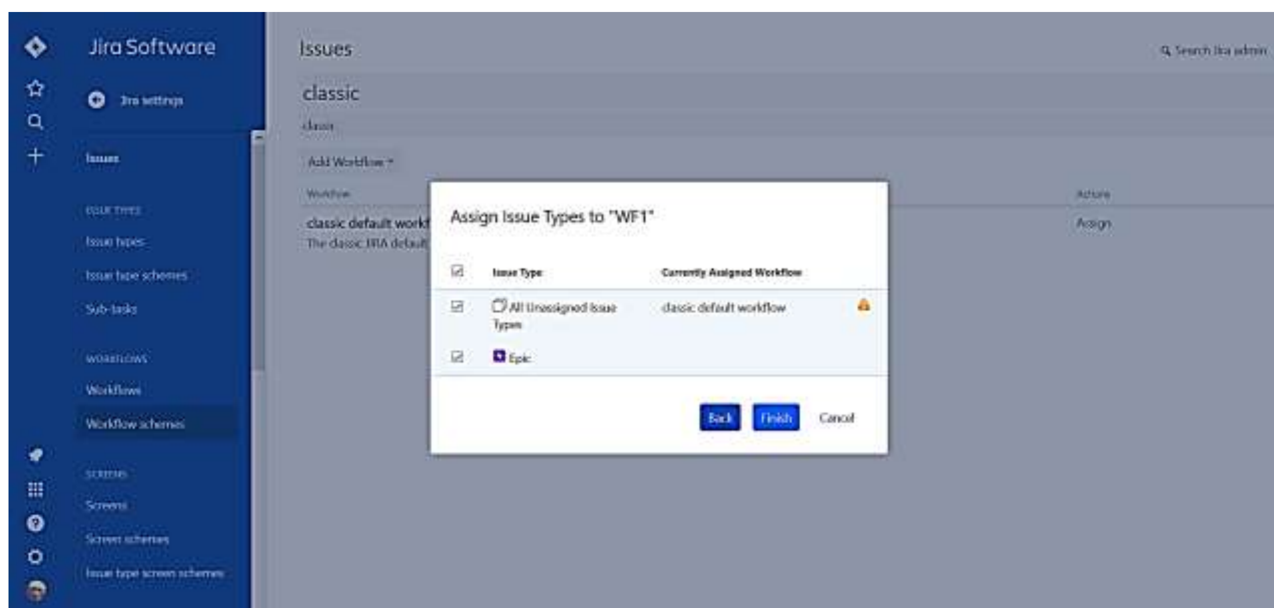


Рисунок 8.24 – Привязка созданного Workflow к проекту

Работа в режиме спринта

Работать над задачами удобнее всего в режиме спринтов. Спринт – это заранее установленный период времени (как правило, две недели), в течение которого должны быть выполнены заранее обговоренные задачи. Работа в спринтах удобна тем, что фронт и сроки работы известны заранее. На рисунке 8.25 показано, как можно создать новый спринт.

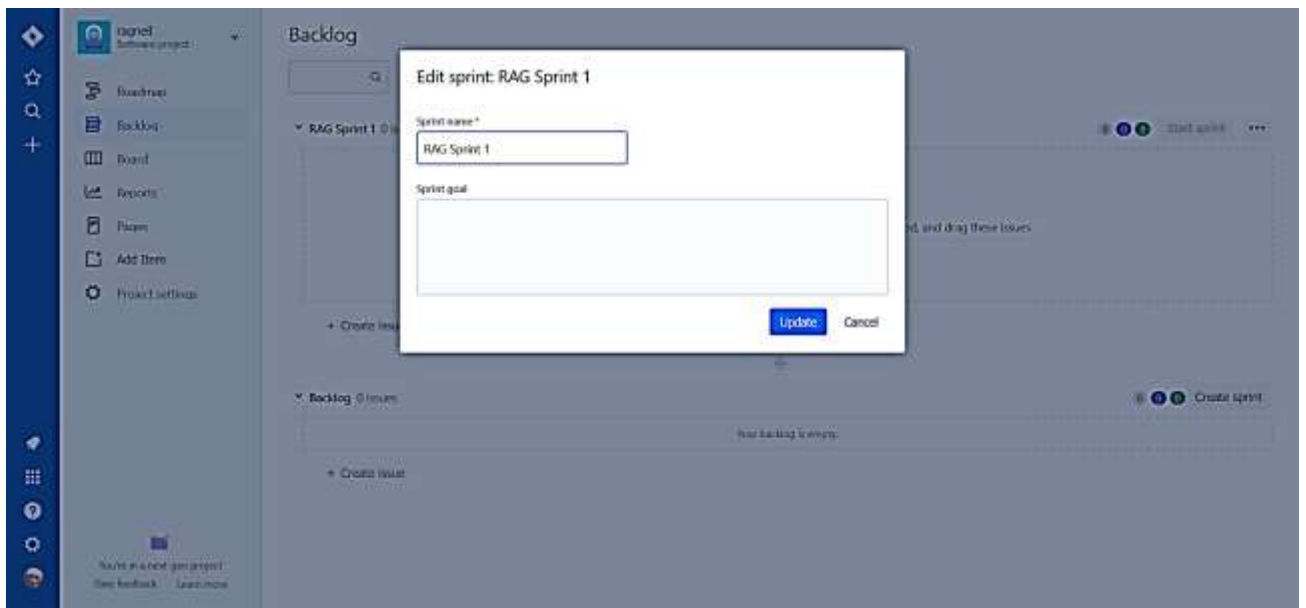


Рисунок 8.25 – Создание спринта

Затем необходимо решить, какие именно задачи войдут в спринт, и раздать их участникам проекта. Пример создания и раздачи новой задачи показан на рисунках 8.26 и 8.27.

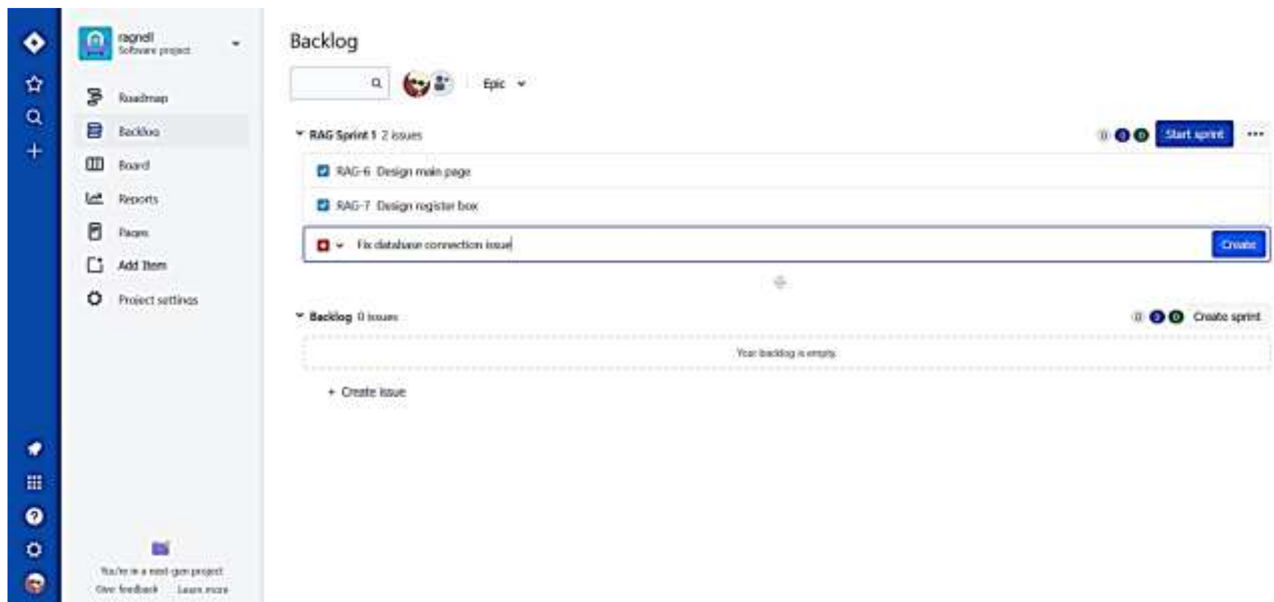


Рисунок 8.26 – Создание задач

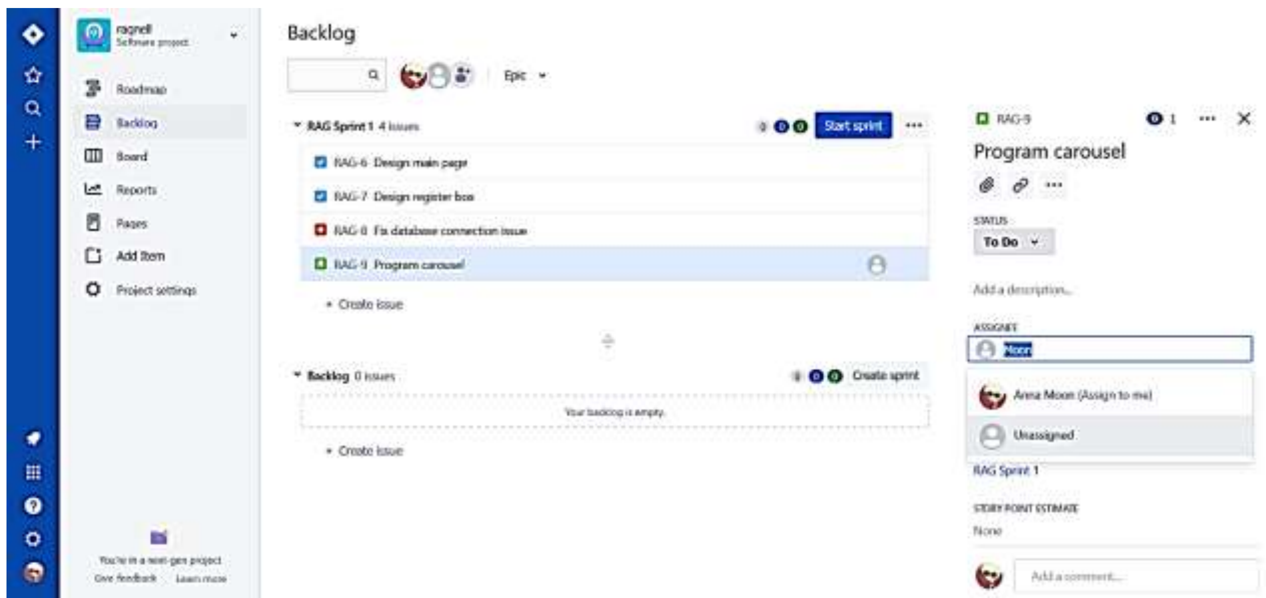


Рисунок 8.27 – Раздача задач

После того, как был определён фронт работы и розданы задачи, можно начинать спринт, как показано на рисунке 8.28.

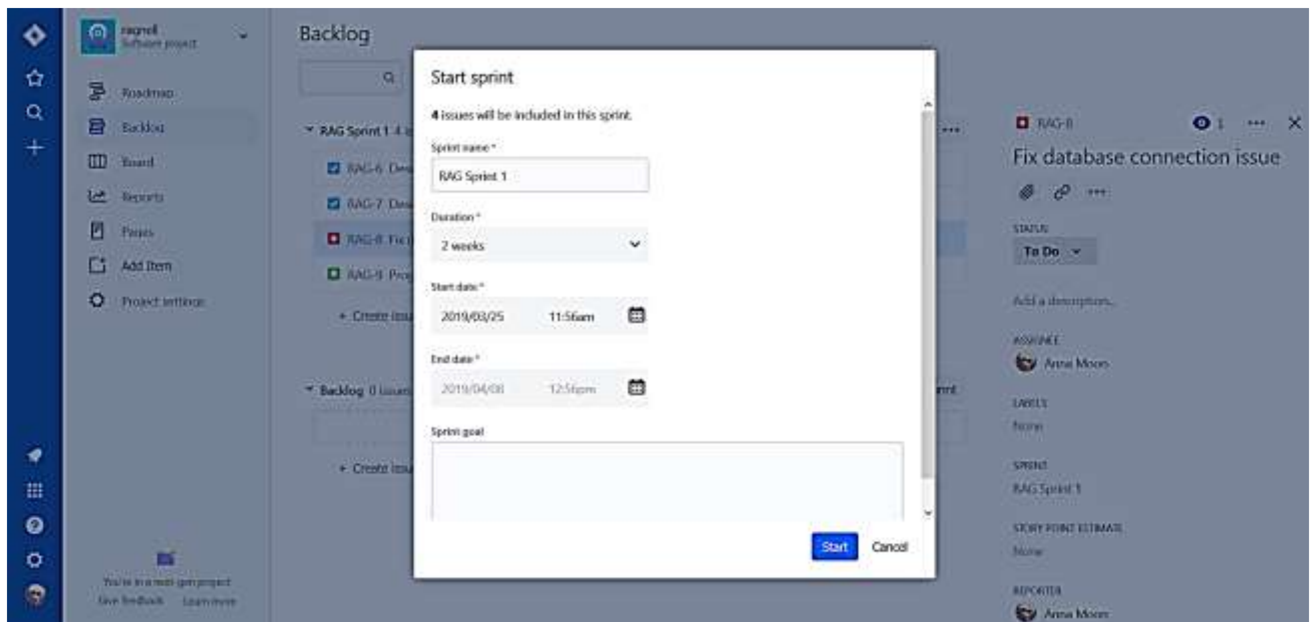


Рисунок 8.28 – Начало спринта

Весь смысл использования систем для отслеживания ошибок заключается в возможности просматривать, каким образом идёт работа над той или иной задачей. Именно поэтому так важно предоставлять отчёты, скриншоты и комментарии о проделанной работе. Пример созданной задачи и все ее поля, доступные для заполнения, показан на рисунке 8.29.

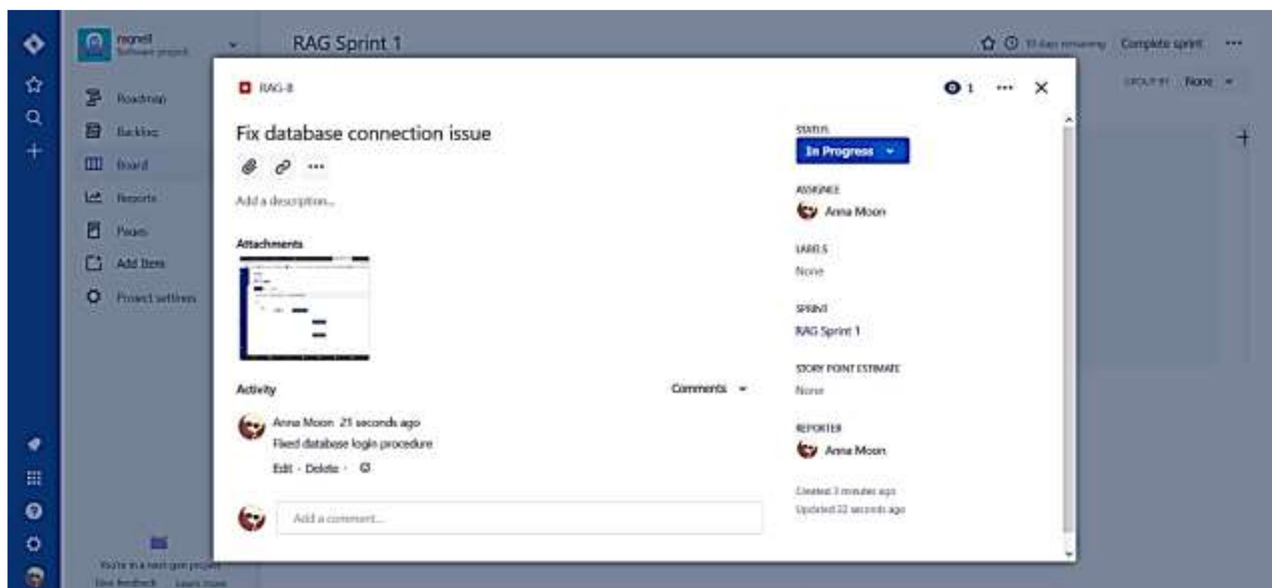


Рисунок 8.29 – Задача в процессе

Доска предоставляет удобный для восприятия обзор всех задач, включённых в спринт. Сразу видно, над какими задачами идёт работа, какие ожидают проверки, а за какие ещё даже не взялись. Активный спринт, на котором видна проводимая работа, показан на рисунке 8.30, тогда как на рисунке 8.31 представлен уже законченный спринт со всеми задачами в последней стадии выполнения.

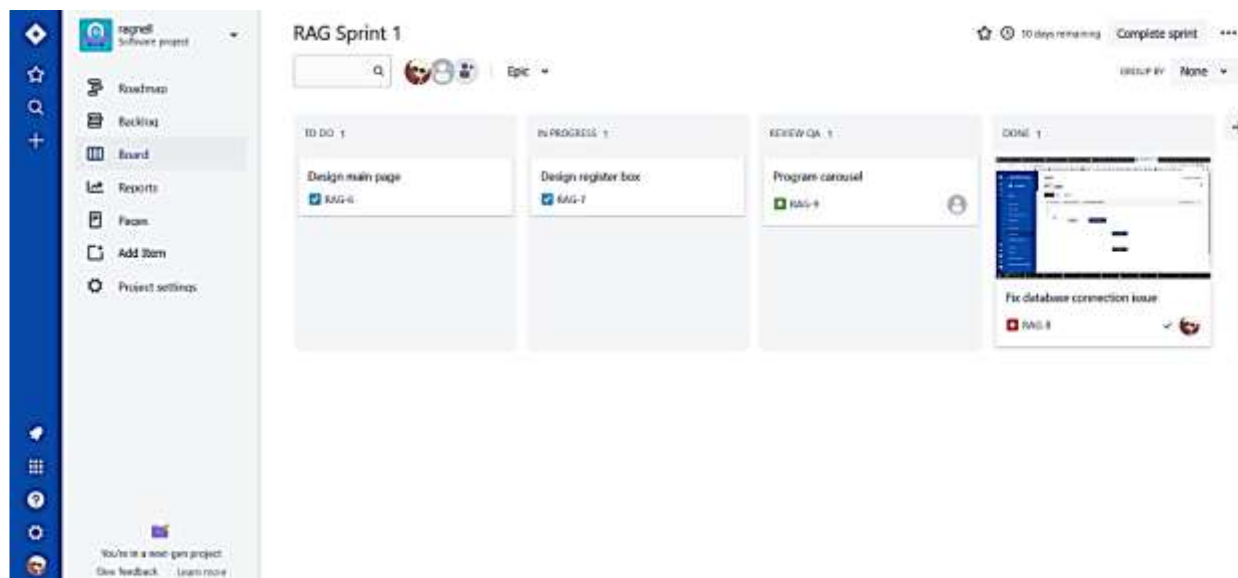


Рисунок 8.30 – Ход работы

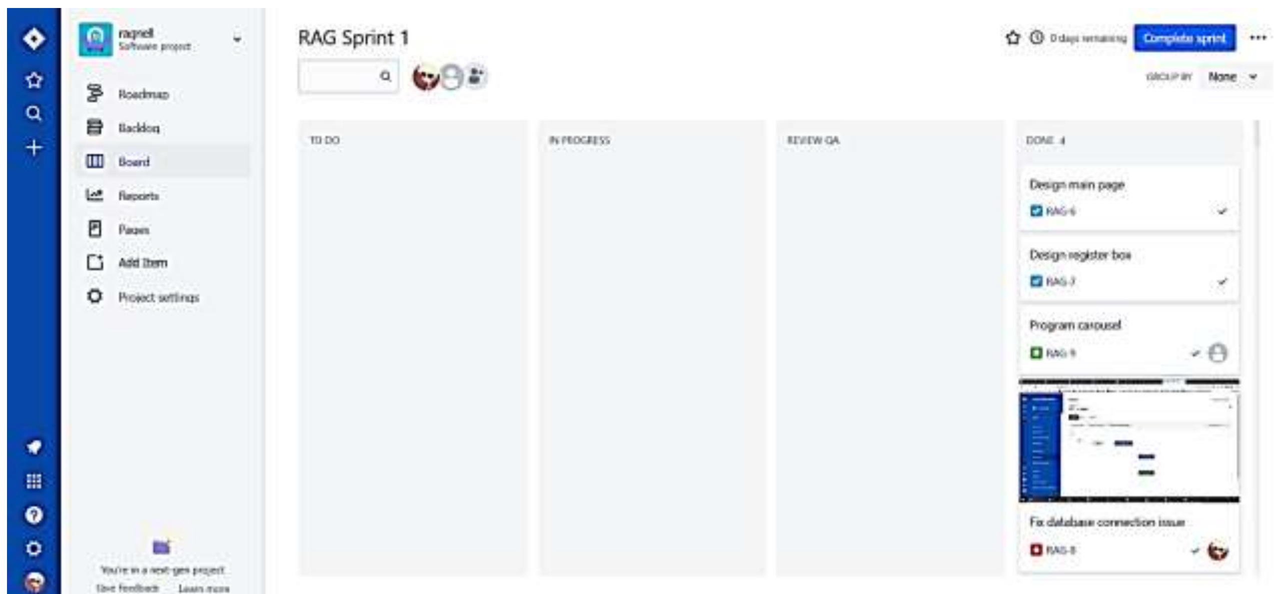


Рисунок 8.31 – Задачи успешно выполнены

После того, как будут выполнены все задачи, или по истечении заданного периода времени, спринт необходимо завершить.

Задание к лабораторной работе

MS Team Foundation Server

a) Установить систему отслеживания ошибок MS Team Foundation Server.

b) Создать в установленной и сконфигурированной системе новый проект для дальнейшего выполнения заданий.

c) Создать на доске новый элемент типа Product Backlog Item.

d) Исследовать систему airport.md в браузере Internet Explorer или Mozilla Firefox на наличие дефектов и занести найденные дефекты в систему MS TFS согласно шаблону отчета о дефекте. Если дефекты не были найдены, ниже представлен список существующих дефектов, которые необходимо занести в систему багтрекинга.

   1) Невозможно зайти на страницу “Air Tickets” из Home Page – “Top Headlines”.
   2) При неверном заполнении формы “Address your question” сообщение об ошибке выводится на отдельной странице.
   3) Буквы румынского алфавита некорректно отображаются в разделе “Noutaţi”.
   4) Значение кнопки “Address your question” отличается в зависимости от веб-браузера.

ZenHub

a) Подключить плагин ZenHub к своему профилю в GitHub. Прикрепить несколько заданий (tickets) на тестирование (позитивное, негативное) к уже готовой лабораторной работе (необязательно по дисциплине тестирования. В случае, если загруженных лабораторных нет, загрузите). Не забудьте привязать (assign) задание к себе и занести его в соответствующую колонку рабочего места.
b) По нахождению багов, занесите новые задания, пометив их тегом bug. Ошибочный функционал для лабораторной работы можно прописать специально.
c) Разрешите внесенный bug, отпишитесь об этом в задании и закройте его. Не забудьте занести задание в секцию "Done".

Atlassian Jira

a) Зарегистрироваться в системе отслеживания ошибок Atlassian Jira.
b) Создать 4 новые роли для проекта.
c) Создать свой собственный workflow из 4÷5 состояний.
d) Исследовать вебсайт airport.md на наличие багов и занести найденные баги (4÷5 штук) в список задач для предстоящего спринта. Начать спринт.

Контрольные вопросы

a) Объясните понятие "система отслеживания ошибок". Почему эти системы удобно использовать в менеджменте проектов?
b) Какие типы проектов существуют в системе Team Foundation Server от Microsoft?
c) Какие виды досок поддерживает система Atlassian Jira?
d) В чем основное отличие системы ZenHub от других система отслеживания ошибок?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. BEIZER B. – Black-box testing: techniques for functional testing of software and systems.
2. КАНЕР С., ФОЛК Д., НГУЕН Е. К. – Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. – К.: Издательство «ДиаСофт», 2001. – 544 с.
3. ОРЛОВ С. Технологии разработки программного обеспечения: Учебник. - СПб.: Питер, 2002.- 464 с.
4. СИНИЦЫН С. В., НАЛЮТИН Н. Ю. – Верификация программного обеспечения. Курс лекций МГУ. – М., 2006. – 158 с.
5. СТЕПАНЧЕНКО И. В. – Методы тестирования программного обеспечения. Учебное пособие ВГТУ. – Волгоград, 2006. – 74 с.
6. Software-testing Инструменты автоматизации тестирования. [электронный ресурс]. – Режим доступа: http://software-testing.ru/library/testing/mobile-testing/2735-mobile-autotesting
7. Official Documentation UiAutomator library by Google, [электронный ресурс] – Режим доступа: https://developer.android.com/training/testing/ui-automator#java
8. Автоматизация тестирования UiAutomator [электронный ресурс]. – Режим доступа: https://habr.com/en/company/intel/blog/205864/
9. Test Ui for multiple Apps [электронный ресурс]. – Режим доступа: https://developer.android.com/training/testing/ui-testing/uiautomator-testing
10. Example for UiAutomator test [электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/android/android_ui_testing.htm
11. Software-testing Инструменты автоматизации тестирования. [электронный ресурс]. – Режим доступа: http://software-testing.ru/library/testing/mobile-testing/2735-mobile-autotesting
12. Official Documentation UiAutomator library by Google, [электронный ресурс] – Режим доступа: https://developer.android.com/training/testing/ui-automator#java
13. Автоматизация тестирования UiAutomator [электронный ресурс]. – Режим доступа: https://habr.com/en/company/intel/blog/205864/
14. Test Ui for multiple Apps [электронный ресурс]. – Режим доступа: https://developer.android.com/training/testing/ui-testing/uiautomator-testing
15. Example for UiAutomator test [электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/android/android_ui_testing.htm

16. Amber, Тестирование нагрузки веб-сервера при помощи apache Jmeter [электронный ресурс].-Режим доступа: https://www.8host.com/blog/testirovanie-nagruzki-veb-servera-pri-pomoshhi-apache-jmeter/
17. TrueGraph, Как создать нагрузочный тест с помощью apache Jmeter [электронный ресурс]. - Режим доступа**:** https://habr.com/ru/post/165159/
18. Msyu, Простой нагрузочный тест с apache Jmeter, [электронный ресурс]. - Режим доступа**:** https://habr.com/en/post/84190/
19. Федор, Работа с базой из тест-плана Jmeter, [электронный ресурс]. - Режим доступа: http://blackdef.blogspot.com/2016/10/jmeter.html
20. Jmeter: нагрузочное тестирование базы данных, [электронный ресурс]. - Режим доступа: https://blog.soft-industry.com/ru/jmeter-databases-load-testing-part-i/
21. p_lab, Логирование Apache Jmeter в БД Oracle в режиме on-line, [электронный ресурс]. - Режим доступа**:** https://habr.com/ru/company/performance_lab/blog/200164/

ПРИЛОЖЕНИЯ

Приложение А

Форма документа “Отчёт о проблеме”

НАЗВАНИЕ КОМПАНИИ ________________________________ ОТЧЁТ О ПРОБЛЕМЕ № __

ПРОГРАММА ___________________ ВЫПУСК _____ ВЕРСИЯ _____

ТИП ОТЧЕТА (1 – 6) __ СТЕПЕНЬ ВАЖНОСТИ (1 – 3) __ ПРИЛОЖЕНИЯ (Д / Н) __

ТИП ОТЧЕТА	СТЕПЕНЬ ВАЖНОСТИ	ПРИЛОЖЕНИЯ
1 - Ошибка кодирования	1 - Блокирующая	Если да, какие:
2 - Ошибка проектирования	2 - Критическая	
3 - Предложение	3 - Значительная	
4 - Расхождение с документацией	4 - Незначительная	
5 - Взаимодействие с аппаратурой	5 - Тривиальная	
6 - Вопрос		

ПРОБЛЕМА ____

МОЖЕТЕ ЛИ ВЫ ВОСПРОИЗВЕСТИ ПРОБЛЕМНУЮ СИТУАЦИЮ? (Д / Н) __

ПОДРОБНОЕ ОПИСАНИЕ ПРОБЛЕМЫ И КАК ЕЕ ВОСПРОИЗВЕСТИ __________________

ПРЕДЛАГАЕМОЕ ИСПРАВЛЕНИЕ (НЕОБЯЗАТЕЛЬНО) ___________________________

ОТЧЕТ ПРЕДСТАВЛЕН СОТРУДНИКОМ________ ДАТА __ / __ / __

Следующие графы предназначены только для разработчиков

ФУНКЦИОНАЛЬНАЯ ОБЛАСТЬ __________ ОТВЕТСТВЕННОЕ ЛИЦО __________,

КОММЕНТАРИИ ____

СОСТОЯНИЕ (1 – 2) __ ПРИОРИТЕТ (1 – 5) __

СОСТОЯНИЕ	ПРИОРИТЕТ	
1 – Открыто	1 - Высокий	4 - Ниже среднего
2 – Закрыто	2 - Выше среднего	5 - Низкий
	3 - Средний	

РЕЗОЛЮЦИЯ (1 – 9) __ ИСПРАВЛЕННАЯ ВЕРСИЯ _____

1 - Рассматривается	4 - Отложено	7 - Отозвано составителем
2 - Исправлено	5 - Соответствует проекту	8 - Нужна дополнительная информация
3 - Не воспроизводится	6 - Не может быть исправлено	9 - Не согласен с предложением

РАССМОТРЕНО ____________ ДАТА __ / __ / __

ПРОКОНТРОЛИРОВАНО ____________ ДАТА __ / __ / __

СЧИТАТЬ ОТЛОЖЕННЫМ (Д / Н) __

Приложение Б

Информация по оформлению *“Отчёта о проблеме”*

Отчет об ошибке — это технический документ, описывающий ошибку/дефект, найденную в процессе тестирования ПО**.** В связи с этим необходимо отметить, что язык описания проблемы должен быть техническим. Должна использоваться правильная терминология при использовании названий элементов пользовательского интерфейса, действий пользователя и полученных результатах.

Требования к обязательным полям отчёта

Обязательными полями являются: краткое описание (*Bug Summary*), степень важности (*Severity*), приоритет (Priority), шаги к воспроизведению проблемы (*Steps to reproduce*) и т. д. Также необходимо прикрепить файлы (скриншоты, тестовые файлы и др.), которые показывают дефект наглядно, а также использованные для тестирования файлы.

Тип отчёта

В графе “Тип отчёта” указывается тип обнаруженной проблемы.

а) Ошибка кодирования. Программа ведёт себя не так, как должна, по мнению тестировщика. Например, если программа утверждает, что 2+2=3, то это явная ошибка кодирования. Программист же в ответ на отчёт о такой ошибке вполне может написать. Соответствует проекту.

b) Ошибка проектирования. Программа соответствует проектной документации, в определённом вопросе тестировщик с этой документацией не согласен. Так особенно часто случается с элементами пользовательского интерфейса. На отчёте данного типа программист не может написать **Соответствует проекту**, если он считает, что проект верен, тогда он пишет **Не согласен с предложением**.

c) Предложение. Отчёт такого типа не означает, что в программе что-то не так. В нём описывается идея, реализация которой, по мнению тестировщика, может улучшить программу.

d) Расхождение с документацией. Программа ведёт себя не так, как описано в руководстве или интерактивной справки. В этом случае в отчёте следует указать, в каком именно документе и на какой странице найдено несоответствие. При этом в отчёте вовсе не утверждается, что ошибка именно в документации, а не в самой программе. Отчёты о расхождении с документацией обязательно должны совместно рассматриваться программистом и автором документации. О функциях программы, которые вообще нигде не описаны, также следует составлять отчёты данного типа.

e) Взаимодействие с аппаратурой. Проблемы этого рода связаны с неудачным взаимодействием программы и определённого вида аппаратного обеспечения. Если причина неудачи заключается в неисправности устройства, отчёт о ней составлять не нужно. Однако если

программа не может работать ни с одной платой или устройством конкретного типа — это уже проблема, которую следует документировать.

f) Вопрос. Программа делает что-то, чего тестировщик не ожидает или не понимает. Отчёт – вопрос стоит составить при любых сомнениях. Если они окажутся основанными на действительной ошибке, программист её исправит. Если же программист откажется исправить ошибку или его объяснения не покажется вам достаточно разумным, можно будет составить отчёт об ошибке проектирования.

Степень важности (**Severity**) — это атрибут, характеризующий влияние дефекта на работоспособность приложения.

В этой графе тестировщик указывает, насколько, по его мнению, серьёзна выявленная проблема.

К сожалению, для определения степени важности проблемы не существует строго критерия. Бейзер, например, предлагает шкалу от 1 (незначительная ошибка, например грамматическая) до 10 (фатальная, вызывающая сбой в других системах, войны, убийства и т. д.). Однако при этом Бейзер не считает значительными недостатки, которые вызывают у пользователя раздражение или заставляют его терять время.

Приоритет (**Priority**) — это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Можно сказать, что это инструмент менеджера по планированию работ. Чем выше приоритет, тем быстрее нужно исправить дефект.

Градация степени важности ошибки (Severity)

1 Блокирующая (Blocker) Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.

2 Критическая (Critical) Критическая ошибка, неправильно работающая ключевая бизнес-логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.

3 Значительная (Major) Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.

4 Незначительная (Minor) Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.

5 Тривиальная (Trivial) Тривиальная ошибка, не касающаяся бизнес-логики приложения, плохо воспроизводимая проблема, малозаметная по средствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

Приложения

К отчету о найденной ошибке можно приложить дискету с тестовыми данными или программу, эмулирующую действия пользователя, при которых проявляется данная ошибка. Можно приложить распечатки, копии экрана или собственные дополнительные пояснения. Проще говоря, все, что поможет программисту разобраться в ситуации и понять вашу точку зрения, следует передать ему вместе с отчетом и перечислить в графе “**Приложения”** чтобы эти материалы случайно не затерялись.

Проблема

В этой графе суть проблемы формулируется очень коротко – в одной-двух строчках. Но при этом описание должно быть и достаточно информативным, чтобы прочитавший его сотрудник смог сразу составить себе четкое представление о проблеме. Именно по нему он будет искать нужный отчет, если захочет возвратиться к нему повторно. Кроме того, следует иметь в виду, что в сводных перечнях ошибок, как правило, будут присутствовать всего несколько полей: Номер отчета, Степень важности, возможно Тип отчета и Проблема.

Если по этому короткому описанию проблема покажется менее серьезной, чем есть на самом деле, существует риск, что руководитель проигнорирует отчет. Но и слишком сгущать краски тоже нельзя, иначе вы прослывете паникёром.

В графе “**Проблема**” не следует рассказывать, как воспроизвести ошибку. Примером хорошего описания может быть такая строчка: “Сбой программы при попытке сохранения файла под не допустимым именем”.

Можете ли вы воспроизвести проблемную ситуацию

Ответом может быть: **Да, Нет** или **Не всегда**. Если с повторением ситуации возникли сложности, лучше отложить составление отчета до тех пор, пока дело не прояснится: либо вы убедитесь, что не знаете, как ее воспроизвести (и напишете **Нет**), либо поймете, что она носит нерегулярный характер (и напишите **Не всегда**). В последнем случае описать способ воспроизведения ситуации нужно особенно тщательно, указав, при каких обстоятельствах ошибка проявляется, а при каких – нет. Следует помнить, что, если написать в отчете **Да** или **Не всегда**, программист может попросить продемонстрировать описанную ситуацию, и если вы не сможете этого сделать, то зря потратите его время и потеряете доверие. С другой стороны, отчет о проблеме, которую невозможно воспроизвести, программист с большой вероятностью просто отложит, пока не появятся дополнительные отчеты.

Подробное описание проблемы и как ее воспроизвести

Прежде всего, следует подробно написать, в чем состоит проблема, и если это не очевидно, то почему вы считаете, что что-то не в порядке. Опишите все шаги и симптомы, все подробности, включая сообщения об ошибке. В этом разделе отчета лучше предоставить программисту избыточную информацию, чем написать слишком мало.

Составляя описание, тестировщик часто обнаруживает, что не знает точно, при каких условиях проявляется ошибка. Лучше увидеть это сразу и протестировать программу еще немного, чем давать программисту повод усомниться в вашей аккуратности.

Если же воспроизвести ошибку не удается даже после многих попыток, но при этом вы абсолютно уверены, что видели ее, составьте о ней максимально подробный отчет. Хороший программист сможет ее найти по вашему описанию, проанализировав программный код. Опишите все сообщения об ошибках, расскажите, что пытались делать. Но никогда не игнорируйте проблему только потому, что она не производится.

Предлагаемое исправление

Эта графа отчета не является обязательной. Если решение проблемы очевидно или, наоборот, у вас нет конкретного предложения, оставьте ее пустой.

Но не стоит пренебрегать ею, если вы знаете, как исправить найденный недостаток программы. Особенно это касается пользовательского интерфейса: программист может не исправить его просто потому, что не сможет быстро придумать, как это сделать. В то же время неудачный текст на экране или неудобное расположение элементов формы будут исправлены очень быстро, если предложить программисту готовый вариант решения.

Отчёт представлен сотрудником

Обязательно укажите здесь свою фамилию. Если у программиста возникнут вопросы, он должен знать, к кому обратиться. А анонимные отчеты часто вообще игнорируются.

Дата

В этой графе следует указать дату обнаружения проблемы. Это не дата написания отчета и не дата ввода его в компьютер. Поскольку программисты не всегда меняют номер версии программы после внесения в нее очередных изменений (иногда просто забывая это делать), указанная в отчете дата поможет идентифицировать версию программы, к которой он относится.

Функциональная область

В этой графе указывается, к какой категории относится выявленная проблема. Их полный перечень должен быть единым, чтобы во всех отчетах названия функциональных областей были одинаковыми. Кроме того, их не должно быть слишком много, и они должны быть очень чётко определены. Десяток функциональных областей часто является оптимальным количеством.

Поручено

В этой графе должно быть указано, кто из сотрудников отвечает за решение описанной проблемы. Как правило, это решает руководитель проекта, который передаёт отчет конкретному программисту. Тестировщик или даже руководитель группы тестирования не должен решать, кто конкретно должен внести исправления.

Комментарии

Если отслеживание ошибок и их исправления не автоматизировано, а ведётся на бумаге, эта графа используется программистом. Он может коротко записать, почему отчет отложен или как решена проблема.

В многопользовательских системах отслеживания ошибок данное поле используется гораздо более эффективно. Прежде всего, оно может быть любой длины, и каждый, кто имеет доступ к отчету, может внести собственный комментарий. Для исправления сложных ошибок или решения спорных проблем иногда может потребоваться целая дискуссия с несколькими участниками. Свое мнение может высказать программист, один или несколько тестировщиков, члены группы технической поддержки, авторы документации, менеджер по маркетингу, руководитель проекта и др. Это быстрый и очень эффективный способ обмена информацией и мнениями, гораздо более эффективный, чем электронная почта. Некоторые опытные сотрудники групп тестирования считают это поле базы данных одним из самых важных.

Состояние

В поле **Состояние** только что написанного отчета записывается **Открыто.** Поле исправления ошибки или принятия решения, не требующего дальнейшей работы с отчетом, значение этого поля изменяется на **Закрыто**.

В некоторых компаниях используются три варианта состояния вопроса: **Открыто, Закрыто и Решено**. Программисты ищут в базе данных отчеты по состоянию **Открыто**, а тестировщики по состоянию **Решено**. В нашей системе программисты ищут отчеты по резолюции **Рассматривается**, а тестировщики - по состояния **Открыто** с любыми резолюциями, кроме **Рассматривается**. Обе эти системы логически эквивалентны, но у каждой из них есть убежденные сторонники.

Градация Приоритета дефекта (Priority)

1 Высокий (High) Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.

2 Средний (Medium) Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.

3 Низкий (Low) Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.

Порядок исправления ошибок по их приоритетам:

High -> Medium -> Low

Приложение В

Пример спецификации требований

Введение

a) Цели

Реализованный программный продукт Program Sort представляет собой приложение для сортировки массива чисел. Программный продукт должен включать пользовательский интерфейс с полями ввода и вывода массива, кнопкой сортировки. Необходимо наличие текстовых файлов, содержащих исходный и отсортированный массивы.

b) Соглашения о терминах

ПО – программный продукт
Program Sort – программа для сортировки массива чисел по алгоритму «Вставка»
Program Sort Interface – интерфейс для ввода исходного массива и вывода отсортированного массива.

c) Ссылки на источники:

«Ссылки на алгоритм сортировки вставкой»

Общее описание

a) Видение продукта:

Данное ПО является прикладной программой для преподавателей и студентов для сортировки массива чисел по методу сортировки вставкой. ПО состоит из пользовательского интерфейса, а также создаваемого текстового файла для хранения данных.

b) Функциональность продукта:
Данное ПО реализует сортировку вещественных чисел в диапазоне от -32678 до 32677. Время обработки массива зависит от размера исходного массива, а также от того на сколько он не отсортирован.

c) Классы и характеристики пользователей
Преподаватели ВУЗа и студенты специальности информационных технологий дневной и заочной форм обучения.

Функциональность системы *(функциональные требования)*

Требование 1
Исполняющий файл имеет имя Program_Sort и формат .exe

Требование 2
Тестовый файл имеет имя Sort_file и формат .txt

Требование 3
Тестовой файл создается после первого успешного запуска программы.

Требование 4

Тестовой файл обновляется при каждом успешном запуске программы.

Требование 5

Текстовой файл программы размещается в той директории, где сохранен исполняющий файл программы.

Требование 6

Program Sort Interface содержит поле для ввода массива, кнопку «Sort», поле для вывода отсортированного массива.

Требование 7

Поле для ввода массива вмещает 50 символов.

Требование 8

Символами являются числа массива, а также запятые, разделяющие их.

Требование 9

Сообщение об ошибке появляется в следующих случаях:

- в поле для ввода введено более 50 символов и нажата кнопка «Sort»;
- в поле для ввода помимо требуемых символов введены буквы и другие знаки и нажата кнопка «Sort»;
- в поле для ввода введено число менее – 32678, и нажата кнопка «Sort»;
- в поле для ввода введено число более 32677 и нажата кнопка «Sort»;
- в поле для ввода не введено ни одного символа и нажата кнопка «Sort»;
- в поле для ввода введен пробел между двумя числами.

Пометка: при синхронизации данных случаев сообщение об ошибке также должно появляться.

Требование 10

После нажатия кнопки «Sort» в поле для вывода должен появиться отсортированный массив в формате «число 1», «число 2» и так далее.

Требование 11

При успешном завершении программы текстовой файл содержит:

- в первой строке неотсортированный массив;
- во второй строке – отсортированный.

Требование 12

Массив в поле для ввода в Program Sort Interface равен массиву в первой строке текстового файла после успешного завершения работы программы.

Требование 13

Массив в поле для вывода в Program Sort Interface равен массиву во второй строке текстового файла после успешного завершения работы программы.

Требование 14

После закрытия сообщения об ошибке поле для ввода массива очищается автоматически.

Написание тестовых сценариев (test-case)

Тестовые сценарии (тесты, test-cases)– последовательность действий тестера в ПО для проверки его качества или качества его функциональной части. Тестовые сценарии удобнее записывать в таблице. В ней должны быть указаны в колонках следующие пункты:

a) Название теста.
b) Описание теста.
c) Шаг (номер шага).
d) Описание шага (для каждого шага).
e) Ожидаемый результат (для каждого шага).

Также при необходимости можно указывать сложность теста, его приоритет, комментарии, а также предусловия и постусловия.

Пример написания тестового сценария для покрытия некоторых требований ПО Program Sort:

C:\Users\acer\
Desktop\Пример тес

Приложение Г

Пример составления отчета об ошибке:

Описание бага: при закрытии сообщения об ошибке поле для ввода массива не очищается автоматически.

Шаги к воспроизведению:

Номер шага	Описание шага	Ожидаемый результат
Шаг 1	Запустить Program Sort	Программа запущена без ошибок
Шаг 2	В поле для ввода ввести: 51 символ	51 символ должны быть введен
Шаг 3	Нажать кнопку "Sort"	Появилось сообщение об ошибке
Шаг 4	Закрыть сообщение об ошибке	Сообщение об ошибке закрыто. Поле для ввода массива очистилось автоматически.
Шаг 5	Проверить что поле для ввода очищено автоматически	Поле для ввода очищено

Результат шага 5: после закрытия сообщения об ошибке поле для ввода осталось с введенным до этого массовом (51 символ).

Серьезность: Незначительная (Minor)

Приоритет: Низкий (Low)

Файлы: «скриншот до закрытия сообщения об ошибке»

«скриншот после закрытия сообщения об ошибке»