

Федеральное государственное автономное образовательное учреждение высшего
образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

Тестирование программного обеспечения

Проверил:
Сентерев Ю. А. _____
«_____» _____ 2020 г.

Выполнил:
Студент группы Р3455
Федюкович С. А. _____

Оценка _____

Санкт-Петербург
2020

Цель работы

Целью данной лабораторной работы является изучение методологий и овладение навыками проектирования тестов.

В ходе выполнения работы будут получены навыки составления тестовых случаев, а также навыки работы в составе инспекционной группы с подготовкой итогового отчета о выявленных проблемах.

Задачи

1. Ознакомиться с теоретическими сведениями по методам тестирования.
2. В соответствии с выбранным заданием(тестовым случаем), подготовить тесты по методикам стратегии "черного ящика"(«белого ящика»).
3. Тесты свести в таблицу.
4. Выполнить тестирование. Занести в таблицу результаты
5. Сделать вывод о роли тестирования с использованием стратегии "черного ящика"(«белого ящика») и возможностях его применения. Сформулировать его достоинства и недостатки.

Ход Работы

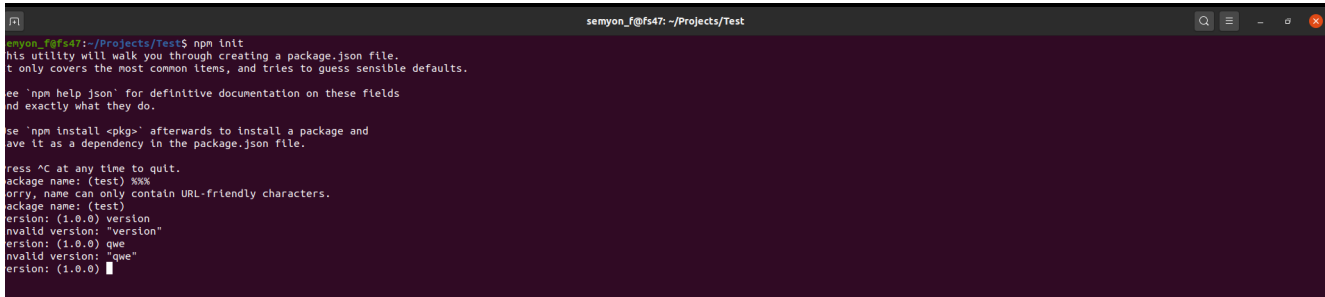
1. С теоретическими сведения ознакомился, готов к выполнению лабораторной работы.
2. Для тестирования была выбрана программа NPM(Node Package Manager) — это менеджер пакетов для программной среды Node.js в виде *CLI*. Данная программа написана на языке программирования JavaScript и является очень популярной в среде разработчиков Node.js и имеет открытый исходный код, поэтому для тестирования была выбрана стратегия "белого ящика". Составим тесты и занесём их в таблицу.

3. Изучив код программы, был составлен следующий список тестов:

Название теста	Тестовый сценарий	Тестовые данные	Ожидаемый результат
Создание модуля с получением ошибок о некорректном названии и версии	В пустой папке запустить создание модуля командой <i>npm init</i> , ввести в название символы %, на этапе ввода версии модуля ввести произвольные символы без точек и цифр	нет	Программа выдаёт ошибку о некорректном названии и версии
Успешное создание модуля с авто заполнением полей	В пустой папке запустить создание модуля командой <i>npm init</i> , ничего не вводить на вопросы <i>npm</i> , завершить создание модуля	нет	Программа успешно создаёт модуль и <i>package.json</i> файл
Установка модуля при некорректном <i>package.json</i> файле	В созданном из предыдущего теста модуле, отредактировать <i>package.json</i> файл, получив некорректный <i>JSON</i> , и установить любой модуль при помощи команды <i>npm install</i>	Успешно созданный модуль	Программа выдаёт ошибку о невозможности установить модуля по причине неисправности <i>package.json</i> файла
Успешная установка модуля	В созданном из прошлого теста модуле установить любой модуль при помощи команды <i>npm install</i>	Успешно созданный модуль	Программа успешно устанавливает модуль, добавляя его в <i>package.json</i>
Успешное удаление модуля	В созданном из прошлого теста модуле и установленном в нём дополнительном модуле удалить этот установленный модуль при помощи команды <i>npm remove</i>	Успешно созданный модуль и установленном к нему дополнительным модулем	Программа успешно удаляет модуль, убирая его из <i>package.json</i>

Таблица 1: План тестирования

4. Для процесса тестирования была создана отдельная папка, в которой и будет происходить запуск тестовой программы. Так же каждый тест подразумевает проверку на то, создавала ли программа лишние файлы или дела ли то, чего не должна. В первом тесте запускается создание модуля и вводятся недопустимые символы создание:



```
semyon_f@fs47: ~/Projects/Test$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

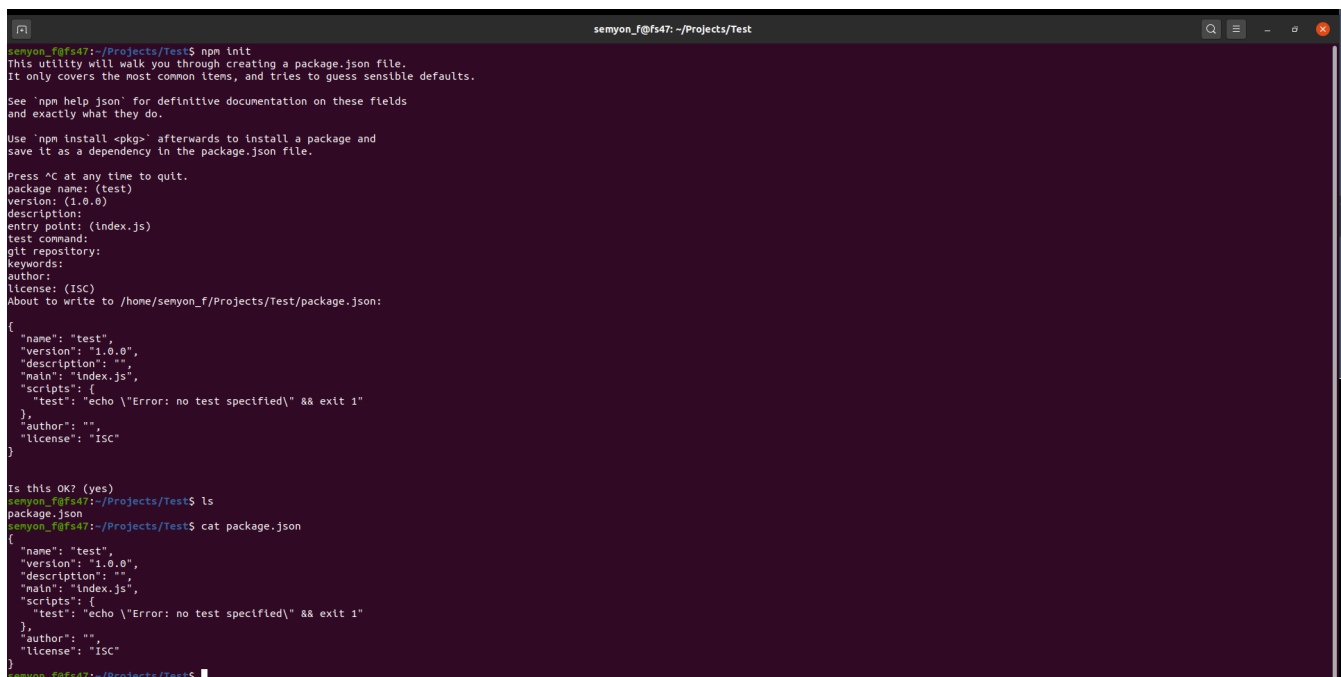
See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (test) %%%
Error: name can only contain URL-friendly characters.
package name: (test)
version: (1.0.0) version
Invalid version: "version"
version: (1.0.0) qwe
Invalid version: "qwe"
version: (1.0.0) %
```

Рис. 1: Создание модуля с получением ошибок о некорректном названии и версии

В следующем тесте запускается создание модуля и не вводятся никакие данные для создания модуля:



```
semyon_f@fs47: ~/Projects/Test$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

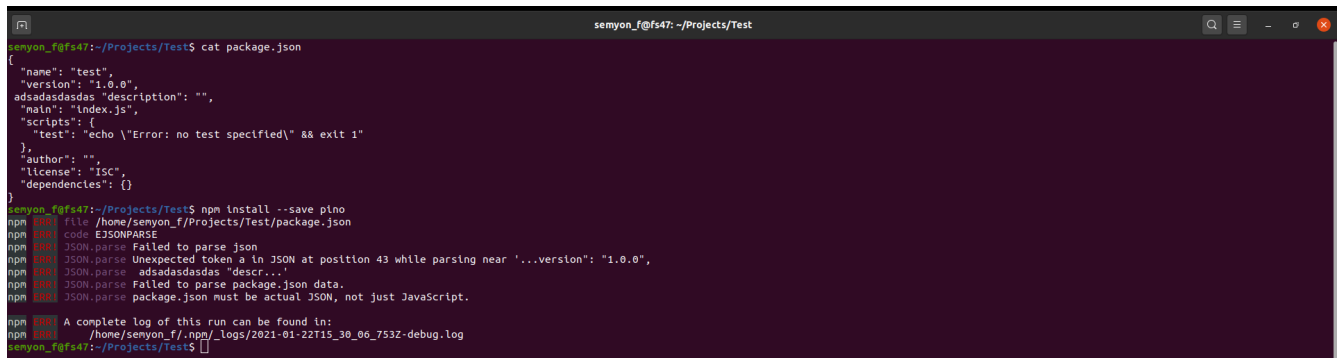
See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (test)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/semyon_f/Projects/Test/package.json:
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
Is this OK? (yes)
semyon_f@fs47: ~/Projects/Test$ ls
package.json
semyon_f@fs47: ~/Projects/Test$ cat package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
semyon_f@fs47: ~/Projects/Test$
```

Рис. 2: Успешное создание модуля с авто заполнением полей

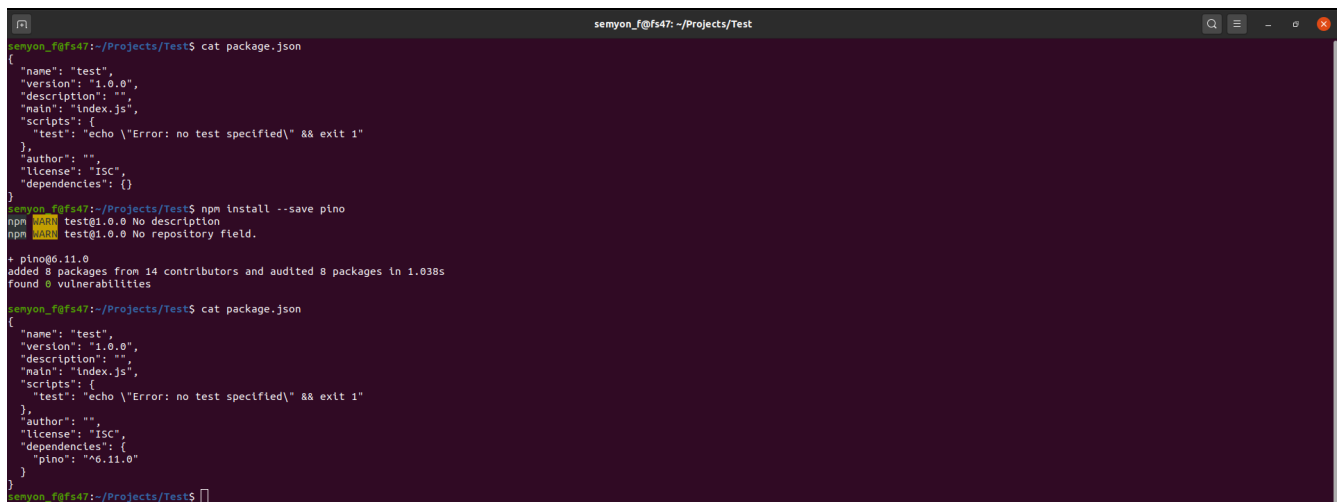
В данном тесте проверяется возможность установить дополнительный модуль с неисправным *package.json* файлом:



```
semyon_f@fs47: ~/Projects/Test
semyon_f@fs47:~/Projects/Test$ cat package.json
{
  "name": "test",
  "version": "1.0.0",
  adsadasdasdas "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\Error: no test specified\\ && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {}
}
semyon_f@fs47:~/Projects/Test$ npm install --save pino
npm ERR! file /home/semyon_f/Projects/Test/package.json
npm ERR! code EJSONPARSE
npm ERR! JSON.parse Failed to parse json
npm ERR! JSON.parse Unexpected token a in JSON at position 43 while parsing near '...version": "1.0.0",
npm ERR! JSON.parse adsadasdasdas "descr...'
npm ERR! JSON.parse Failed to parse package.json data.
npm ERR! JSON.parse package.json must be actual JSON, not just Javascript.
npm ERR! A complete log of this run can be found in:
npm ERR! /home/semyon_f/.npm/_logs/2021-01-22T15_30_06_753Z-debug.log
semyon_f@fs47:~/Projects/Test$
```

Рис. 3: Установка модуля при некорректном *package.json* файле

В последующем тесте проверяется возможность успешно установить дополнительный модуль:

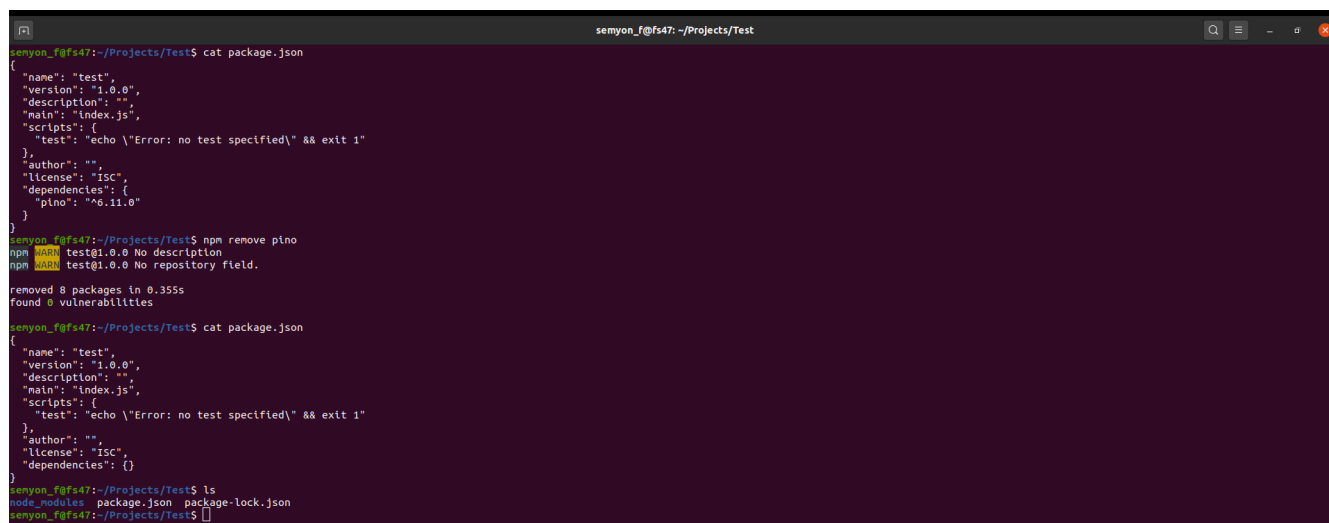


```
semyon_f@fs47:~/Projects/Test$ cat package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\Error: no test specified\\ && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {}
}
semyon_f@fs47:~/Projects/Test$ npm install --save pino
npm WARN test@1.0.0 No description
npm WARN test@1.0.0 No repository field.

+ pino@6.11.0
added 8 packages from 14 contributors and audited 8 packages in 1.038s
found 0 vulnerabilities
semyon_f@fs47:~/Projects/Test$ cat package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\Error: no test specified\\ && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "pino": "^6.11.0"
  }
}
semyon_f@fs47:~/Projects/Test$
```

Рис. 4: Успешная установка модуля

В последнем тесте проверяется возможность успешно удалить дополнительный модуль:



```
semyon_f@fs47:~/Projects/Test$ cat package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "pino": "^6.11.0"
  }
}
semyon_f@fs47:~/Projects/Test$ npm remove pino
npm WARN test@1.0.0 No description
npm WARN test@1.0.0 No repository field.
removed 8 packages in 0.355s
found 0 vulnerabilities
semyon_f@fs47:~/Projects/Test$ cat package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {}
}
semyon_f@fs47:~/Projects/Test$ ls
node_modules package.json package-lock.json
semyon_f@fs47:~/Projects/Test$
```

Рис. 5: Успешное удаление модуля

Итоговые результаты тестирования представлены в таблице ниже:

Название теста	Фактический результат	Ожидаемый результат
Создание модуля с получением ошибок о некорректном названии и версии	Программа выдаёт ошибку о некорректном названии и версии	Программа выдаёт ошибку о некорректном названии и версии
Успешное создание модуля с автозаполнением полей	Программа успешно создаёт модуль и <i>package.json</i> файл	Программа успешно создаёт модуль и <i>package.json</i> файл
Установка модуля при некорректном <i>package.json</i> файле	Программа выдаёт ошибку о невозможности установить модуль по причине неисправности <i>package.json</i> файла	Программа выдаёт ошибку о невозможности установить модуль по причине неисправности <i>package.json</i> файла
Успешная установка модуля	Программа успешно устанавливает модуль, добавляя его в <i>package.json</i>	Программа успешно устанавливает модуль, добавляя его в <i>package.json</i>
Успешное удаление модуля	Программа успешно удаляет модуль, убирая его из <i>package.json</i>	Программа успешно удаляет модуль, убирая его из <i>package.json</i>

Таблица 2: Результаты тестирования

Всего было проведено 5 тестов, все из которых были пройдены без ошибок. Процент ошибок равен нулю.

5. Был сделан вывод о роли тестирования по стратегии белого ящика: тестирование и использование этой стратегии позволяет обнаружить ошибки, специфичные для конкретной реализации программы, которые могли бы быть обнаружены при тестировании стратегией черного ящика. Тестовые сценарии будут меняться только при изменении интерфейса модуля, такие тесты не будут меняться при изменении деталей реализации ПО. С другой стороны, тестирование по стратегии черного ящика может начать разработку тестовых сценариев одновременно с разработкой ПО.

Вывод

В ходе лабораторной работы я провёл тестирование программы, изучил методологии тестирования ПО и выполнил все поставленные задачи. Лабораторную работу считаю выполненной.

Используемая литература

1. Гленфорд Майерс, Том Баджетт, Кори Сандлер. Искусство тестирования программ, 3-е издание—М.: «Диалектика», 2015
2. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем — СПб.: Питер, 2004
3. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений — Киев: ДияСофт, 2001
4. Винниченко И. Автоматизация процессов тестирования. — СПб, «Питер», 2018
5. Котляров В. П., Коликова Т. В. Основы тестирования программного обеспечения — СПб, Бином. Лаборатория знаний, 2006