**Валидация, Верификация и Тестови план**

Автор: Александър Ивайлов Иванов  
Факултетен номер: 501217031  
Група: 42 гр.

Факултет: ФКСТ

Специалност: ИТИ

**Проект: FKST.VVSS.2020.SHAKE-256(Output bits 512)**

**Ревизионен лист**

|  |  |  |
| --- | --- | --- |
| **Релийз No.** | **Дата** | **Описание** |
| Rev. 0 | 30/04/20 | Попълване на общата информация на тестовия план |
| Rev. 1 | 1/05/20 | Добавяне на функционални тестове |
| Rev. 2 | 2/05/20 | Добавяне на модулни тестове |
| Rev. 3 | 3/05/20 | Изготвяне на Матрица за проследяване на изискванията |
| Rev. 4 | 4/05/20 | Изготвяне на доклад на модулни и фунционални тестове |
| Rev. 5 | 5/05/20 | Оформление на плана за тестване |
|  |  |  |
|  |  |  |

**Валидация, Верификация и Тестови план**

**СЪДЪРЖАНИЕ**

Page #

1.0 Обща информация 1-1

1.1 Цел 1-1

1.2 Обхват 1-1

1.3 Общ преглед на системата 1-1

1.5 Съкращения 1-1

2.0 Оценка на тестването 2-1

2.1 Матрица за проследяване на изискванията (Requirements Traceability Matrix) 2-1

2.2 Критерии за оценка на тест 2-1

3.0 Описание на тестовите случаи 3-1

3.1 Функционални тестове 3-1

3.1.1 Тест за хеширане на стринг, въведен преди стартиране на програмата 3-1

 *Тестови / функционални връзки* 3-1

 *Средства за контрол* 3-1

 *Входни данни* 3-1

 *Изходни данни* 3-1

 *Процедура за тестване* 3-1

3.1.2 Тест за хеширане на стринг, въведен след стартиране на програмта 3-2

 *Тестови / функционални връзки* 3-2

 *Средства за контрол* 3-2

 *Входни данни* 3-2

 *Изходни данни* 3-2

 *Процедура за тестване* 3-2

3.1.3 Тест за хеширане на файл, ако файлът не е намерен 3-3

 *Тестови / функционални връзки* 3-3

 *Средства за контрол* 3-3

 *Входни данни* 3-3

 *Изходни данни* 3-3

 *Процедура за тестване* 3-3

3.1.4 Тест за хеширане на файл, ако файлът не е в правилния формат 3-4

 *Тестови / функционални връзки* 3-4

 *Средства за контрол* 3-4

 *Входни данни* 3-4

 *Изходни данни* 3-4

 *Процедура за тестване* 3-4

3.1.5 Тест за хеширане на файл, ако файлът е в правилния формат 3-5

 *Тестови / функционални връзки* 3-5

 *Средства за контрол* 3-5

 *Входни данни* 3-5

 *Изходни данни* 3-5

 *Процедура за тестване* 3-5

3.1.6 Тест за хеширане на файл, който е празен 3-6

 *Тестови / функционални връзки* 3-6

 *Средства за контрол* 3-6

 *Входни данни* 3-6

 *Изходни данни* 3-6

 *Процедура за тестване* 3-6

3.1.7 Тест за хеширане на файл, съдържащ само един ред 3-7

 *Тестови / функционални връзки* 3-7

 *Средства за контрол* 3-7

 *Входни данни* 3-7

 *Изходни данни* 3-7

 *Процедура за тестване* 3-7

3.2 Модулни (Unit) тестове 3-8

3.2.1 TEST001 (test()) 3-8

 *Средства за контрол* 3-8

 *Входни данни* 3-8

 *Изходни данни* 3-8

 *Покритие на теста* 3-8

3.2.1 TEST002 (test()) 3-8

 *Средства за контрол* 3-8

 *Входни данни* 3-8

 *Изходни данни* 3-8

 *Покритие на теста* 3-8

3.2.1 TEST003 (test()) 3-9

 *Средства за контрол* 3-9

 *Входни данни* 3-9

 *Изходни данни* 3-9

 *Покритие на теста* 3-9

4.0 Покритие на всички тестове 3-9

# Обща информация

## 1.1 Цел

Целта на системата е да хешира подаденото съобщение, използвайки алгоритъма SHAKE-256, като хеширащото съобщение трябва да има изходен формат от 512 bits(64 bytes) или 128 символа.

## 1.2 Обхват

Обхватът на системата включва хеширане на всякакъв вид въведено съобщение или съобщение от файл само в текстови формат(разширение - .txt).

## 1.3 Общ преглед на системата

Системата хешира съобщение, подадено й по 3 възможни начина – предварително въведено съобщение, съобщение въведено след стартиране на програмата и съобщение, прочетено от текстови файл(.txt).

**1. Клас TestClass**

В него създаваме инстации на класовeте, които искаме да тестваме.

**2. Клас FkstHashString**

В него имаме конструктор, който приема съобщението за хеширане, и метод hashString, който подава съобщението на хеширащия метод и връща хешираното съобщение.

**3. Клас FkstHashStringConsole**

В него имаме метод hashStringConsoleInput, който приема съобщение от клавиатурата и го подава на хеширащия метод и връща хешираното съобщение.

**4. Клас FkstHashFile**

В него имаме конструктор, който приема файла за хеширане, проверяваме дали е текстови и съществува, извличаме съдържанието му, подава го на хеширащия метод и връща хешираното съобщение.

**5. Клас CreateHashShake256**

В него се хешират съобщенията, използвайки алгоритъма SHAKE-256(Output bits 512).

**6.** **Клас Shake256Utilities**

В него подготвяме подаденото съобщение за хеширане.

## 1.5 Съкращения

# ОЦенка на тестването

## 2.1 Матрица за проследяване на изискванията (Requirements Traceability Matrix)

File: **TraceabilityMatrix\_Opencodez.xls**

## 2.2 Критерии за оценка на тест

Segment 1: Трябва да хешира съобщение, въведено преди стартиране на програмата. Критерии – хешираното съобщение трябва да е във формат 512 бита и да е бъде изведено на конзолата. Всяка заявка трябва да се изпълни за под 2 сек.

Segment 2: Трябва да хешира съобщение, въведено от клавиатурата след стартиране на програмата. Критерии – хешираното съобщение трябва да е във формат 512 бита и да е бъде изведено на конзолата. Всяка заявка трябва да се изпълни за под 2 сек.

Segment 3: Трябва да хешира съобщение, извлечено от файл. Критерии – файлът задължително трябва да е в текстови формат(.txt). Хешираното съобщение трябва да е във формат 512 бита и да е бъде изведено на конзолата. Всяка заявка трябва да се изпълни за под 3 сек. Aко файлът не е в подходящ формат или не е намерен, да се извежда подходящо съобщение за грешка.

# Описание на тестовите случаи

## 3.1 Функционални тестове

### 3.1.1 [Тест за хеширане на стринг, въведен преди стартиране на програмата]

**Описание -** тестът сравнява дали резултатът, върнат от метода hashString(), е равен на очакавания резултат.

**Елементи:** обект от тип FkstHashString, метод за сравняване assertEquals().

### *Тестови / функционални връзки*

TC#001 - Entered a message before executing the program

### *Средства за контрол*

Тестът се изпълнява полу-автоматично.

### *Входни данни*

message = "Test Program"

### *Изходни данни*

Hash Message = 545a5e2db9bd957eb789b44046c0a5a5da9af640d54756b5d24f13d02e0f4a5 64e3980fb0419117a1dc8063b3f5f6ae5bf7122cf95847bf43c0db33c84c24a0d

### *Процедура за тестване*

1. Създаваме test case class – TestHashStringMethod

2. Автоматично ни се генерира test() метод, в който създаваме истанция на класа, които тестваме FkstHashString.

3. Подаваме като параметър на конструктора му съобщението, което искаме да хешираме.

4. Извикваме метода assertEquals(), в който подаваме като параметри очаквания резултат и резултата, който ще ни върне методът hashString() от класа FkstHashString.

5. Стартираме теста – Ctrl + F11.

6. Като резултат ни се изписва, че тестът е успешен – методът е върнал такъв резултат, какъвто очакваме.

### 3.1.2 [Тест за хеширане на стринг, въведен след стартиране на програмта]

**Описание -** тестът сравнява дали резултатът, върнат от метода hashStringConsoleInput(), е равен на очакавания резултат.

**Елементи:** обект от тип FkstHashStringConsole, метод за сравняване assertEquals().

### *Тестови / функционални връзки*

TC#002 - Enter a message after executing the program

### *Средства за контрол*

Тестът се изпълнява ръчно.

### *Входни данни*

message = “Console Input”;

### *Изходни данни*

Hash Message = 1d89828a68af73e4175848c15b216df759bcce591dceb0e88eb4a237d49b64d4cc 8510921d5ce48b40259b652469291bc07541df04c4165909c510eeec14bf2d

### *Процедура за тестване*

1.Създаваме test case class – TestHashStringConsoleInputMethod.

2. Автоматично се генерира test() метод, в който създаваме истанция на класа, които тестваме FkstHashStringConsole.

3. Извикваме метода assertEquals(), в който подаваме като параметри очаквания резултат и резултата, който ще върне методът hashStringConsoleInput() от класа TestHashStringConsoleInput.

4. Стартираме теста – Ctrl + F11.

5. След стартиране на теста, въвеждаме съобщението, което искаме да хешираме.

6. Като резултат ни се изписва, че тестът е успешен – методът е върнал такъв резултат, какъвто очакваме.

### 3.1.3 [Тест за хеширане на файл, ако файлът не е намерен]

**Описание -** тестът сравнява дали резултатът, върнат от метода hashTextFile(), е равен на очакавания резултат.

**Елементи:** обект от тип FkstHashFile, обект от тип File, метод за сравняване assertEquals().

### *Тестови / функционални връзки*

TC#003 – File does not exist

### *Средства за контрол*

Тестът се изпълнява полу-автоматично.

### *Входни данни*

file = “test2.txt”;

### *Изходни данни*

The File not found!

### *Процедура за тестване*

1. Създаваме test case class – TestHashFileMethodWithMissingFile.

2. Автоматично се генерира test() метод, в който създаваме обект от тип File.

3. Подаваме като параметър на конструктора му името на файла, чието съдържание ще хешираме.

4. Създаваме инстанция на класа, които тестваме – FkstHashFile.

5. Подаваме като параметър на конструктора обекта от тип File, който сме създали.

6. Извикваме метода assertEquals(), в който подаваме като параметри очаквания резултат и резултата, който ще ни върне методът hashTextFile() от класа FkstHashFile.

7. Стартираме теста – Ctrl + F11.

8. Като резултат ни се изписва, че тестът не е успешен – методът не е върнал такъв резултат, какъвто очакваме.

* + 1. **3.1.4 [Тест за хеширане на файл, ако файлът не е в правилния формат]**

**Описание -** тестът сравнява дали резултатът, върнат от метода hashTextFile(), е равен на очакавания резултат.

**Елементи:** обект от тип FkstHashFile, обект от тип File, метод за сравняване assertEquals().

* ***Тестови / функционални връзки***

TC#004 - File format is not correct

* ***Средства за контрол***

Тестът се изпълнява полу-автоматично..

* ***Входни данни***

file = “test.docx”;

* ***Изходни данни***

The File is not in text format!

* ***Процедура за тестване***

1. Създаваме test case class – TestHashFileMethodWithInCorrectFormat.

2. Автоматично се генерира test() метод, в който създаваме обект от тип файл.

3. Подаваме като параметър на конструктора му името на файла, чието съдържание ще хешираме.

4. Създаваме инстанция на класа, които тестваме – FkstHashFile.

5. Подаваме като параметър на конструктора обекта от тип файл, който сме създали.

6. Извикваме метода assertEquals(), в който подаваме като параметри очаквания резултат и резултата, който ще ни върне методът hashTextFile() от класа FkstHashFile.

7. Стартираме теста – Ctrl + F11.

8. Като резултат ни се изписва, че тестът не е успешен – методът не е върнал такъв резултат, какъвто очакваме.

* + 1. **3.1.5 [Тест за хеширане на файл, ако файлът е в правилния формат]**

**Описание -** тестът сравнява дали резултатът, върнат от метода hashTextFile(), е равен на очакавания резултат.

**Елементи:** обект от тип FkstHashFile, обект от тип File, метод за сравняване assertEquals().

* ***Тестови / функционални връзки***

TC#005 - File format is correct

* ***Средства за контрол***

Тестът се изпълнява полу-автоматично.

* ***Входни данни***

file = “test.txt”;

* ***Изходни данни***

Hash Message = ad9252183c39cb5a17a0dad0254707b6dd9e69ff894f8cd51fec365409d02a1e8b597 7ae8122349914142d6744763570a38a6d23b7cd399968f451b3c10cc1ba

* ***Процедура за тестване***

1. Създаваме test case class - TestHashFileMethodWithAtLeastTwoLinesFile.

2. Автоматично ни се генерира test() метод, в който създаваме обект от тип файл.

3. Подаваме като параметър на конструктора му името на файла, чието съдържание ще хешираме.

4. Създаваме инстанция на класа, които тестваме – FkstHashFile.

5. Подаваме като параметър на конструктора обекта от тип файл, който сме създали.

6. Извикваме метода assertEquals(), в който подаваме като параметри очаквания резултат и резултата, който ще ни върне методът hashTextFile() от класа FkstHashFile.

7. Стартираме теста – Ctrl + F11.

8. Като резултат ни се изписва, че тестът е успешен – методът е върнал такъв резултат, какъвто очакваме.

* + 1. **3.1.6 [Тест за хеширане на файл, който е празен]**

**Описание -** тестът сравнява дали резултатът, върнат от метода hashTextFile(), е равен на очакавания резултат.

**Елементи:** обект от тип FkstHashFile, обект от тип File, метод за сравняване assertEquals().

* ***Тестови / функционални връзки***

TC#006 - File is empty

* ***Средства за контрол***

Тестът се изпълнява полу-автоматично.

* ***Входни данни***

file = “testFileEmpty.txt”;

* ***Изходни данни***

Hash Message = 46b9dd2b0ba88d13233b3feb743eeb243fcd52ea62b81b82b50c27646ed5762fd75d c4ddd8c0f200cb05019d67b592f6fc821c49479ab48640292eacb3b7c4be

* ***Процедура за тестване***

1. Създаваме test case class - TestHashFileMethodWithEmptyFile.

2. Автоматично ни се генерира test() метод, в който създаваме обект от тип файл.

3. Подаваме като параметър на конструктора му името на файла, чието съдържание ще хешираме.

4. Създаваме инстанция на класа, които тестваме – FkstHashFile.

5. Подаваме като параметър на конструктора обекта от тип файл, който сме създали.

6. Извикваме метода assertEquals(), в който подаваме като параметри очаквания резултат и резултата, който ще ни върне методът hashTextFile() от класа FkstHashFile.

7. Стартираме теста – Ctrl + F11.

8. Като резултат ни се изписва, че тестът е успешен – методът е върнал такъв резултат, какъвто очакваме.

* + 1. **3.1.7 [Тест за хеширане на файл, съдържащ само един ред]**

**Описание -** тестът сравнява дали резултатът, върнат от метода hashTextFile(), е равен на очакавания резултат.

**Елементи:** обект от тип FkstHashFile, обект от тип File, метод за сравняване assertEquals().

* ***Тестови / функционални връзки***

TC#007 - File contains only one line

* ***Средства за контрол***

Тестът се изпълнява полу-автоматично.

* ***Входни данни***

file = “testFileWithOneLine.txt”;

* ***Изходни данни***

Hash Message = a3d3baf91495ad21119744574c27eabfa4c0c6276da19149da021c4ff9645cba90a54 82875c79d53b6ff92fcdef8684553e3441222dbb6128ee32a67ba8e3178

* ***Процедура за тестване***

1. Създаваме test case class - TestHashFileMethodWithOnlyOneLineFile.

2. Автоматично ни се генерира test() метод, в който създаваме обект от тип файл.

3. Подаваме като параметър на конструктора му името на файла, чието съдържание ще хешираме.

4. Създаваме инстанция на класа, които тестваме – FkstHashFile.

5. Подаваме като параметър на конструктора обекта от тип файл, който сме създали.

6. Извикваме метода assertEquals(), в който подаваме като параметри очаквания резултат и резултата, който ще ни върне методът hashTextFile() от класа FkstHashFile.

7. Стартираме теста – Ctrl + F11.

8. Като резултат ни се изписва, че тестът е успешен – методът е върнал такъв резултат, какъвто очакваме.

## 3.2 Модулни (Unit) тестове

### 3.2.1 TEST001 (test())

### Тестът е описан в RTM таблицата. Функционалната му връзка е с Shake256Utilities класа и тества един от методите в самия клас – getMessageBytes(String text), който подготвя байтовете преди хеширане.

### *Средства за контрол*

Тестът се изпълнява полу-автоматично.

### *Входни данни*

Текстово съобщение

### *Изходни данни*

Масив от байтове

### *Покритие на теста*

В пакет shake256:

Класове 50% 1/2  
Методи 25% 1/4  
Редове 18% 3/11

### 3.2.2 TEST002 (test())

### Тестът е описан в RTM таблицата. Функционалната му връзка е със Shake256Utilities класа и тества един от методите в самия клас – getHashBytes(byte[] messageBytes), който хешира самите байтове преди хеширане.

### *Средства за контрол*

Тестът се изпълнява полу-автоматично.

### *Входни данни*

Масив от байтове

### *Изходни данни*

Масив от хеширани байтове

### *Покритие на теста*

В пакет shake256:

Класове 50% 1/2  
Методи 25% 1/4  
Редове 45% 5/11

### 3.2.3 TEST003 (test())

### Тестът е описан в RTM таблицата. Функционалната му връзка е със Shake256Utilities класа и тества един от методите в самия клас – getHashHexMessage(byte[] hashBytes), който хешира самите байтове преди хеширане.

### *Средства за контрол*

Тестът се изпълнява автоматично.

### *Входни данни*

Масив от хеширани байтове

### *Изходни данни*

Хеширано съобщение

### *Покритие на теста*

В пакет shake256:

Класове 50% 1/2  
Методи 25% 1/4  
Редове 63% 7/11

# Покритие на всички тестове

**В пакет shake256:**

Класове 50% 1/2  
Методи 75% 3/4  
Редове 63% 7/11

**В клас Shake256Utilities:**

Класове 100% 1/1  
Методи 100% 3/3  
Редове 100% 7/7