Reference

[1] Chumbley, Alexander. *A Version Control System for Everyone*. 2016, Chumbley, Alexander. A Version Control System for Everyone. dspace.mit.edu/bitstream/handle/1721.1/112825/1014182419-MIT.pdf?sequence=1.

[2] https://www.cumbria.ac.uk/media/university-of-cumbria-website/content-assets/public/vco/documents/recordsmanagement/VersionControl.pdf

[3] http://www.theseus.fi/bitstream/handle/10024/118000/Miettinen_Antti.pdf;jsessionid=3B791BAE631440C64FAB718E4EEBD24F?sequence=1

[4] Gupta, Piyush, and Sandeep Kumar. "A Comparative Analysis of SHA and MD5 Algorithm ."

[5] *Secure Hash Algorithms*, en.wikipedia.org/wiki/Secure_Hash_Algorithms.

[7] Weber, Sandra. "Automatic Version Control System for Distributed Software Development." *Automatic Version Control System for Distributed Software Development*, Sept. 2012.

[8] Koc, Ali, and Abdullah Uz Tansel. "A Survey of Version Control Systems ." *A Survey of Version Control Systems*.

[9] Carlsson, Emil. "Mining Git Repositories, An Introduction to Repository Mining." *Mining Git Repositories, An Introduction to Repository Mining*, 2 Aug. 2013, pp. 1–43.

[10] Git, "git/git · GitHub," GitHub, [Online]. Available: https://github.com/git/git

Appendix A - Source code to “Media Share 1.1 android”

I could complete admin and lecturers uploading features using media share code. Especially I integrated admin panel to the mobile app implementation. Here I choose a mobile application ‘Media Share’, one of famous Learning Management System for this development.

From admin panel, I could complete the assignment uploading part to the system. It works like this. The lecturer can create an assignment for the related course. In an assignment, can include assignment name, date, description and the uploading file. There is an uploading area to upload the related file.

And as for this research, I added history button to see uploaded file content.

When creating an assignment for the first time, lecturer or the course owner is uploading a document. And the system auto takes hashtag for the file content and to the file name. When he re-uploading, the system generates hashtags for new content and for the file name. And then check these hashtags only, to detect the equality.

This is the Android java logic for reading the uploading file, send the read content to generate Hashtag and compare hashtags.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == Activity.RESULT_OK) {
        if (requestCode == PICK_FILE_REQUEST) {
            if (data == null) {
                //no data present
                return;
            }


// Select the file to read.
            Uri selectedFileUri = data.getData();
            selectedFilePath = UOMFilePath.getPath(this, selectedFileUri);

            File file = new File(selectedFilePath);
            currentFileName = file.getName();

// The Content turn to Strings
            fileToString = UOMReading.readDoc(file);
```

// Pass the String to the String Comparison method.

```
            stringComparisonForInstructor(fileToString);

            Log.i(TAG, "Selected File Path:" + selectedFilePath);
            pathList.add(selectedFilePath);
            for (int i = 0; i < pathList.size(); i++) {
                if (i == pathList.size() - 1) {
                    String fileName = new File(pathList.get(i)).getName();
                    fileListName = fileListName + "  " + fileName;
                    /*notificationLabel.setText(fileListName);*/
                }
            }
            hashValueForUploadingString = mdForString(fileToString);
            if (previousAssignmentHashValues != null) {
                String[] sp = previousAssignmentHashValues.split(" ");
                fileValidating(currentFileName, assignmentBackupString,
hashValueForUploadingString, previousAssignmentHashValues);
                assignmentNameCompare();

            } else {
                String previousFileName = "";
                fileValidating(name.getText().toString(),
assignmentBackupString, hashValueForUploadingString, "");
            }

            String fileName = new File(selectedFilePath).getName();
            notificationLabel.setText(fileName););*/
        }
    }
}
```

The logic for the read Files.

```
public static String readDoc(File f) {
    String text = "";
    int read, N = 1024 * 1024;
    char[] buffer = new char[N];

    try {
        FileReader fr = new FileReader(f);
        BufferedReader br = new BufferedReader(fr);

        while (true) {
            read = br.read(buffer, 0, N);
            text += new String(buffer, 0, read);

            if (read < N) {
                break;
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return text;
}
```

// The String comparison method, take the previously stored strings (This has stored as an array and take the new String which has to compare and create another array from that Strings.

```
private void stringComparisonForInstructor(String value) {
    if (savedAssignmentFileString != null) {
        parts = new ArrayList <>(Arrays.asList(value.split("\n")));
        String savedValues = savedAssignmentFileString;// this string value
should comes from the Sever
        ArrayList <String> savedParts = new
ArrayList<>(Arrays.asList(savedValues.split("\n")));

// Prepare hash sets for comparison.
        Set <String> set = new HashSet <String>(savedParts);
        Set <String> partset = new HashSet <>(parts);

        removeAll(parts, set);
    }
}
```

// Prepare hash sets for comparison.

This is a one of famous scenario (Compare two strings an give the difference between them). As an example,

“I have 2 ArrayLists A and B of the same data structure C (hashCode() and equals() overridden). C represents a student's record. The two lists are of the same size and represent new student records and old ones respectively (the students are the same in both the lists, ordering might be different). I wish to keep only those records in A that have been changed. As such, I did: A.removeAll(B)”
As per the Javadocs, this would take each record of A and compare with each record of B, and if it finds both equal, it will remove the record from A. If a record of A is not found to be equal to any record in B, and since all students in A are also in B, it means that that record of A has changed. The problem is that its easily of **n square complexity**.

I have encountered a performance bottleneck in member removeAll (above-mentioned example, defined functionality use in Java or also –dff in GIT) in some instances.

Hence, avoid relying on hidden good properties of specific implementations of List< T > ; Set. Contains() O(1) is a guarantee, use that to bound algorithmic complexity.

I use the following code that avoids useless copies; the intention is that you are scanning a data structure finding irrelevant elements you don't want and adding them to "todel".

For some reason like avoiding concurrent modifications, you are navigating a tree etc..., you cannot remove elements as you are doing this traversal. So, we cumulate them into a HashSet "todel".

In the function, we need to modify "container" in place, since it is typically an attribute of the caller, but using remove (int index) on "container" might induce a copy because of left shift of elements. We use a copy "contents" to achieve this.

Template argument is because during the selection process, I often get subtypes of C, but feel free to use < T > everywhere.

```
public static <T> void removeAll(List <T> container, Set <? extends T>
toDelete) {
    if (toDelete.isEmpty())
        return;
    List <T> contents = new ArrayList <T>(container);
    container.clear();
 // since container contains no duplicates ensure |B| max contains() operations
    int toRemove = toDelete.size();
    for (T elt : contents) {
        if (toRemove == 0 || !toDelete.contains(elt)) {
            container.add(elt);
        } else {
            toRemove--;
        }
    }
    // Show the lists
    System.out.println("First List: " + container + toRemove);
    System.out.println("Second List: " + toRemove);
    HTMLVIewer.setTextOfATextView(comparisonLabel, "added or removed lines."
+ "\n" + container.toString() + "\n\n" + "to Remove:  " + toRemove);
}
```

// Prepare hashtags for comparison.

```
public static String mdForString(String input) {
    try {
```

// MD5 hashtags for comparison.
// MD5 was developed by Professor Ronald L. Rivest in 1994. Its 128 bit (16 byte) message digest makes it a faster implementation than SHA-1. This means that MD5 executes faster.

```
        java.security.MessageDigest md =
java.security.MessageDigest.getInstance("MD5");
        byte[] array = md.digest(input.getBytes("UTF-8"));
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < array.length; i++) {
            sb.append(String.format("%02x", array[i]));
        }
```

```
            return sb.toString();
        } catch (NoSuchAlgorithmException | UnsupportedEncodingException e) {
            return null;
        }
    }
```

// Compare two tags.

```
private boolean compareTag(String uploadingFilePath, ArrayList
savedHashList) {
    String hashValueForLoaclFile = generateMdFiveFlag(uploadingFilePath);
    for (int i = 0; i < savedHashList.size(); i++) {
        if (hashValueForLoaclFile.equals(savedHashList.get(i))) {
            return true;
        }
    }
    return false;
}
```

These are the major steps to validate Lecturer's updates. The lecturer can see his mistakes or what is he going to update before he does his uploading.

Basically, I focus on the major requirement came from their side. Users need accurate and quick results on this. If the system gives valid, correct results but it is performance vice week, it is not a good move. So the researcher always focuses on the performance of the application.

Appendix B - Integrated to “Moodle3.5”

I could integrate the suggested feature using Moodle 3.5 one of famous LMS in the world. It is developed by using PHP. I could attach my development especially thinking about student's functionality. Focusing to reduce uploading mistakes.

// Here I used hashing techniques to track file changes.

No problems with the SHA1. It can use for the track more deeply and performance vice it takes more times than MD5, but I suggest this time is not critical for students parts.

```
public function do_upload()

   {

      $this->multiple_upload();

      if (count($this->upload_errors) > 0) {

         $error = array_map(function ($k, $v) {

            return "$k => $v";

         }, array_keys($this->upload_errors),
array_values($this->upload_errors));

         $this->load->view('upload_form', ['error' => implode('', $error),
'success' => '']);

      }


      if (count($this->upload_data) > 0) {

         $error = "";

         $success = "";



         foreach($this->upload_data as $item){
```

```
            $sha1_hash = sha1_file($item['full_path']);
            $count = $this->count_hash($sha1_hash);
            if($count > 0){
                $error .= "File (".$item['file_name'].") already uploaded<br>";
            }else{
                $item['sha1_hash'] = $sha1_hash;
                $this->handle_file($item);
                $success .= "File (".$item['file_name'].") uploaded
successfully<br>";
        }
         }

        $this->load->view('upload_form', ['error' => $error, 'success' =>
$success]);
    }
  }
  public function handle_file($data)
  {
    $this->db->insert('attempts', [
        'file_name' => $data['file_name'],
        'hash' => $data['sha1_hash'],
        'as_at' => date('Y-m-d H:i A')
    ]);
  }
  private function count_hash($file_hash)
  {
```

```
        $this->db->where('hash', $file_hash);
        $this->db->from('attempts');
        return $this->db->count_all_results();
    }

    public function multiple_upload()
    {
        $this->load->library('upload');
        $number_of_files_uploaded = count($_FILES['userFiles']['name']);

        for ($i = 0; $i < $number_of_files_uploaded; $i++) {
            $_FILES['userfile']['name'] = $_FILES['userFiles']['name'][$i];
            $_FILES['userfile']['type'] = $_FILES['userFiles']['type'][$i];
            $_FILES['userfile']['tmp_name'] =
$_FILES['userFiles']['tmp_name'][$i];
            $_FILES['userfile']['error'] = $_FILES['userFiles']['error'][$i];
            $_FILES['userfile']['size'] = $_FILES['userFiles']['size'][$i];
            $config = array(
              //'file_name'     => <your ouw function to generate random names>,
                'allowed_types' => 'gif|jpg|png|pdf|ppt|doc|docx|txt|xls|xlsx|pptx',
                'max_size' => 10000,
                'overwrite' => FALSE,
                'upload_path' => './uploads/'

            );
            $this->upload->initialize($config);
```

```
                if (!$this->upload->do_upload()) {

                    $this->upload_errors[$_FILES['userFiles']['name'][$i]] =
        $this->upload->display_errors();

                } else {

                    $this->upload_data[] = $this->upload->data();

                }

            }

    }
```

// Using this, the system track changes of the uploading files.

If the user changes one of his uploading files and going to upload all the files, not only the changed one, the system detects only that changed one, and select to upload only that. It gives some benefits. Reduce time-consuming, Save the internet data.

Appendix C - Calculate the speed performance of the MD5 and SHAs.

```
import java.util.UUID;
import org.apache.commons.codec.digest.DigestUtils;
import org.apache.commons.lang.time.StopWatch;
public class Test {
   private static final int TIMES = 1_000_000;
   private static final String UUID_STRING = UUID.randomUUID().toString();
   public static void main(String[] args) {
      System.out.println(generateStringToHash());
      System.out.println("MD5: " + md5());
      System.out.println("SHA-1: " + sha1());
      System.out.println("SHA-256: " + sha256());
      System.out.println("SHA-512: " + sha512());
   }
   public static long md5() {
      StopWatch watch = new StopWatch();
      watch.start();
      for (int i = 0; i < TIMES; i++) {
         DigestUtils.md5Hex(generateStringToHash());
      }
      watch.stop();
      System.out.println(DigestUtils.md5Hex(generateStringToHash()));
      return watch.getTime();
```

```
        }
        public static long sha1() {
            System.out.println(DigestUtils.sha1Hex(generateStringToHash()));
            return watch.getTime();
        }
    public static long sha256() {
System.out.println(DigestUtils.sha256Hex(generateStringToHash()));
            return watch.getTime();
        }
        public static long sha512() {
      System.out.println(DigestUtils.sha512Hex(generateStringToHash()));
            return watch.getTime();
        }
        public static String generateStringToHash() {
            return UUID.randomUUID().toString() + System.currentTimeMillis();
        }
}
```

Several measurements were done. Two groups – one with smaller length string to hash and one with longer. Each group had following variations of generateStringToHash() method:

cached UUID–no extra time should be consumed
cached UUID + current system time – in this case, time is consumed to get system time
new UUID + current system time – in this case, time is consumed for generating the UUID and to get system time

5 Raw results

Five measurements were made for each case an average value calculated. Time is in milliseconds per 1 000 000 calculations. The system is 64 bits Windows 10 with 1 core Intel i7 2.60GHz and 16GB RAM.

generateStringToHash() with: return UUID_STRING;
Data to encode is ~36 characters in length (f5cdcda7-d873-455f-9902-dc9c7894bee0). UUID is cached and time stamp is not taken. No additional time is wasted.

Hash	#1 (ms)	#2 (ms)	#3 (ms)	#4 (ms)	#5 (ms)	Average per 1M (ms)
MD5	649	623	621	624	620	627.4
SHA-1	608	588	630	600	594	604
SHA-256	746	724	741	720	758	737.8
SHA-512	1073	1055	1050	1052	1052	1056.4

Table appendix 1 hashing performance part 1

generateStringToHash() with: return UUID_STRING + System.currentTimeMillis();

Data to encode is ~49 characters in length (*aa096640-21d6-4f44-9c49-4115d3fa69381468217419114*). UUID is cached.

Hash	#1 (ms)	#2 (ms)	#3 (ms)	#4 (ms)	#5 (ms)	Average per 1M (ms)
MD5	751	789	745	806	737	765.6
SHA-1	768	765	694	763	751	748.2
SHA-256	842	876	848	839	850	851
SHA-512	1161	1152	1164	1154	1163	1158.8

Table appendix 2 hashing performance part 2

generateStringToHash() with: return UUID.randomUUID().toString() + System.currentTimeMillis();
Data to encode is ~49 characters in length (*1af4a3e1-1d92-40e7-8a74-7bb7394211e01468216765464*).

New UUID is generated on each calculation so time for its generation is included in total time.

4	#1 (ms)	#2 (ms)	#3 (ms)	#4 (ms)	#5 (ms)	Average per 1M (ms)
MD5	1505	1471	1518	1463	1487	1488.8
SHA-1	1333	1309	1323	1326	1334	1325
SHA-256	1505	1496	1507	1498	1516	1504.4
SHA-512	1834	1827	1833	1836	1857	1837.4

Table appendix 3 hashing performance part 3

generateStringToHash() with: return UUID_STRING + UUID_STRING;

Data to encode is ~72 characters in length (*57149cb6-991c-4ffd-9c98-*

d823ee8a61f757149cb6-991c-4ffd-9c98-d823ee8a61f7). UUID is cached and time stamp is not taken. No additional time is wasted.

Hash	#1 (ms)	#2 (ms)	#3 (ms)	#4 (ms)	#5 (ms)	Average per 1M (ms)
MD5	856	824	876	811	828	839
SHA-1	921	896	970	904	893	916.8
SHA-256	1145	1137	1241	1141	1177	1168.2
SHA-512	1133	1131	1116	1102	1110	1118.4

Table appendix 4 hashing performance part 4

generateStringToHash() with: return UUID_STRING + UUID_STRING + System.currentTimeMillis();

Data to encode is ~85 characters in length (*759529c5-1f57-4167-b289-899c163c775e759529c5-1f57-4167-b289-899c163c775e1468218673060*). UUID is cached.

Hash	#1 (ms)	#2 (ms)	#3 (ms)	#4 (ms)	#5 (ms)	Average per 1M (ms)
MD5	1029	1035	1034	1012	1037	1029.4
SHA-1	1008	1016	1027	1007	990	1009.6
SHA-256	1254	1249	1290	1259	1248	1260
SHA-512	1228	1221	1232	1230	1226	1227.4

Table appendix 5 hashing performance part 5

generateStringToHash() with: final String randomUuid = UUID.randomUUID().toString();

return randomUuid + randomUuid + System.currentTimeMillis();

Data to encode is ~85 characters in length (2734b31f-16db-4eba-afd5-121d0670ffa72734b31f-16db-4eba-afd5-121d0670ffa71468217683040). New UUID is generated on each calculation so time for its generation is included in total time.

Hash	#1 (ms)	#2 (ms)	#3 (ms)	#4 (ms)	#5 (ms)	Average per 1M (ms)
MD5	1753	1757	1739	1751	1691	1738.2
SHA-1	1634	1634	1627	1634	1633	1632.4
SHA-256	1962	1956	1988	1988	1924	1963.6
SHA-512	1909	1946	1936	1929	1895	1923

Table appendix 6 hashing performance part 6

4.1 Aggregated results

Results from all iterations are aggregated and compared in the table below. There are 6 main cases. They are listed below and referenced in the table below:

Case 1 – 36 characters length string, UUID is cached

Case 2 – 49 characters length string, UUID is cached and system time stamp is calculated each iteration

Case 3 – 49 characters length string, new UUID is generated on each iteration and system time stamp is calculated each iteration

Case 4 – 72 characters length string, UUID is cached

Case 5 – 85 characters length string, UUID is cached and system time stamp is calculated each iteration

Case 6 – 85 characters length string, new UUID is generated on each iteration and system time stamp is calculated each iteration

All times below are per 1 000 000 calculations:

Hash	Case 1 (ms)	Case 2 (ms)	Case 3 (ms)	Case 4 (ms)	Case 5 (ms)	Case 6 (ms)
MD5	627.4	765.6	1488.8	839	1029.4	1738.2
SHA-1	604	748.2	1325	916.8	1009.6	1632.4
SHA-256	737.8	851	1504.4	1168.2	1260	1963.6
SHA-512	1056.4	1158.8	1837.4	1118.4	1227.4	1923

Table appendix 7 hashing performance part 7

Compare results

Some conclusions of the results based on two cases with short string (36 and 49 chars) and longer string (72 and 85 chars).

SHA-256 is faster with 31% than SHA-512 only when hashing small strings. When the string is longer SHA-512 is faster with 2.9%.

Time to get system time stamp is ~121.6 ms per 1M iterations.

Time to generate UUID is ~670.4 ms per 1M iterations.

SHA-1 is fastest hashing function with ~587.9 ms per 1M operations for short strings and 881.7 ms per 1M for longer strings.

MD5 is 7.6% slower than SHA-1 for short strings and 1.3% for longer strings.

SHA-256 is 15.5% slower than SHA-1 for short strings and 23.4% for longer strings.

SHA-512 is 51.7% slower that SHA-1 for short strings and 20% for longer.

Hash sizes

Important data to consider is hash size that is produced by each function:

MD5 produces 32 chars hash – *5f3a47d4c0f703c5d83265c3669f95e6*
SHA-1 produces 40 chars hash – *2c5a70165585bd4409aedeea289628fa6074e17e*
SHA-256 produces 64 chars hash –
b6ba4d0a53ddc447b25cb32b154c47f33770d479869be794ccc94dffa1698cd0
SHA-512 produces 128 chars hash –
54cdb8ee95fa7264b7eca84766ecccde7fd9e3e00c8b8bf518e9fcff52ad061ad28cae49ec3a09144ee8f342666462743718b5a73215bee373ed6f3120d30351

Purpose of use

In specific case this research was made for hashed string will be passed as API request. It is constructed from API Key + Secret Key + current time in seconds. So if API Key is something like 15-20 chars, Secret Key is 10-15 chars and time is 10 chars, total length of string to hash is 35-45 chars. Since it is being passed as request param it is better to be as short as possible.

Select hash function

Based on all data so far SHA-256 is selected. It is from secure SHA-2 family. It is much faster than SHA-512 with shorter stings and it produces 64 chars hash.

The conclusion of the sub research

The current post gives a comparison of MD5, SHA-1, SHA-256 and SHA-512 cryptographic hash functions. Important is that comparison is very dependant on specific implementation (Apache Commons Codec), the specific purpose of use (generate a secure token to be sent with API call). It is good MD5 and SHA-1 to be avoided as they are compromised and not secure. If their speed for given context is several times faster than secure SHA-2 ones and security is not that much important they can be chosen though. When choosing cryptographic hash function everything is up to a context of usage and benchmark tests for this context is needed.

Appendix D – SUS Calculation.

To calculate a score between 0 and 100 for the product:

1. Convert SUS responses to numbers, 1 for “Strongly Disagree”, and 5 for “Strongly Agree”.
2. For odd-numbered questions, subtract 1 from the response.
3. For even-numbered questions, subtract the response from 5.
4. Add the scores from each question and multiply the total by 2.5.
5. Remember to present the numbers as a SUS score, not a percentage.

If there are a small numbers of participants, consider calculating a confidence interval around the SUS score. This can help you understand the variability.

Here’s an overview of how the scores should measure:

80.3 or higher is an A. People love your site and will recommend it to their friends

68 or thereabouts gets you a C. You’re doing OK but could improve

51 or under gets you a big fat F. Make usability your priority now and fix this fast.

Table appendix 8 evaluate SUS calculation for Introducing system

SUS CALCULATION FOR THE RESEARCH AFETR DONE THE SUGGESTED FEATURES - VERSION CONTROLLING LOGICS APPLY IN LEARNING MANAGEMENT SYSTEMS													
Question No	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Sum**		**multiply by 2.5**
2nd Step	3	4	4	4	4	4	3	3	3	0	32	2.5	80
1st Step	4	1	5	1	5	1	4	2	4	5			
	Agree	Strongly Disagree	Strongly Agree	Strongly Disagree	Strongly Agree	Strongly Disagree	Agree	Disagree	Agree	Strongly Disagree			
2nd Step	3	2	3	2	2	2	2	3	2	2	23	2.5	57.5
1st Step	4	3	4	3	3	3	3	2	3	3			
	Agree	Neutral	Agree	Neutral	Neutral	Neutral	Neutral	Disagree	Neutral	Neutral			
2nd Step	3	3	3	3	3	3	3	3	3	3	30	2.5	75
1st Step	4	2	4	2	4	2	4	2	4	2			
	Agree	Disagree	Agree	Disagree	Agree	Disagree	Agree	Disagree	Agree	Disagree			
2nd Step	3	3	3	2	3	3	2	3	3	2	27	2.5	67.5
1st Step	4	2	4	3	4	2	3	2	4	3			
	Agree	Disagree	Agree	Neutral	Agree	Disagree	Neutral	Disagree	Agree	Neutral			
2nd Step	3	3	2	0	3	2	3	3	3	2	24	2.5	60
1st Step	4	2	3	5	4	3	4	2	4	3			
	Agree	Disagree	Neutral	Strongly Agree	Agree	Neutral	Agree	Disagree	Agree	Neutral			
2nd Step	3	2	3	4	3	1	1	2	3	1	23	2.5	57.5
1st Step	4	3	4	5	4	4	2	3	4	4			
	Agree	Neutral	Agree	Strongly Agree	Agree	Agree	Disagree	Neutral	Agree	Agree			
2nd Step	4	3	3	3	4	4	4	4	4	3	36	2.5	90
1st Step	5	2	4	2	5	1	5	1	5	2			
	Strongly Agre	Disagree	Strongly Agree	Disagree	Strongly Agree	Strongly Disagree	Strongly Agree	Strongly Disagree	Strongly Agree	Disagree			
2nd Step	3	3	3	2	3	2	3	3	3	3	28	2.5	70
1st Step	4	2	4	3	4	3	4	2	4	2			
	Agree	Disagree	Agree	Neutral	Agree	Neutral	Agree	Disagree	Agree	Disagree			
2nd Step	3	3	3	3	3	3	4	4	3	3	32	2.5	80
1st Step	4	2	4	2	4	2	5	1	4	2			
	Agree	Disagree	Agree	Disagree	Agree	Disagree	Strongly Agree	Strongly Disagree	Agree	Disagree			
2nd Step	3	2	3	0	3	1	1	2	3	1	19	2.5	47.5
1st Step	4	3	4	5	4	4	2	3	4	4			
	Agree	Neutral	Agree	Strongly Agree	Agree	Agree	Disagree	Neutral	Agree	Agree			
2nd Step	3	3	3	3	3	3	4	4	3	3	32	2.5	80
1st Step	4	2	4	2	4	2	5	1	4	2			
	Agree	Disagree	Agree	Disagree	Agree	Disagree	Strongly Agree	Strongly Disagree	Agree	Disagree			
2nd Step	4	4	3	0	3	2	3	2	4	3	28	2.5	70
1st Step	5	1	4	5	4	3	4	3	5	2			
	Strongly Agre	Stongly Disagree	Agree	Strongly Agree	Agree	Disagree	Agree	Nutral	Strongly Agree	Disagree			
2nd Step	4	3	3	3	3	3	3	2	4	3	31	2.5	77.5
1st Step	5	2	4	2	4	2	4	3	5	2			
	Strongly Agre	Disagree	Agree	Strongly Disagree	Agree	Disagree	Agree	Nutral	Strongly Agree	Disagree			
				Strongly Agree (5 point), Agree (4 point), Neutral (3 point), Disagree (2 point) and Strongly Disagree (1 point)									912.5
													912.5 / 13
													70.19