# A SOFTWARE TOOL TO FACILITATE AUTOMATE CREATION OF VIRTUAL INSPECTION TEAMS AND INSPECTION PERFORMANCE EVALUATION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Zhou Lu

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

April 2016

Fargo, North Dakota

North Dakota State University
Graduate School

**Title**

A SOFTWARE TOOL TO FACILITATE AUTOMATE CREATION OF VIRTUAL INSPECTION TEAMS AND INSPECTION PERFORMANCE EVALUATION

**By**

Zhou Lu

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

SUPERVISORY COMMITTEE:

Dr. Gursimran Walia
Chair

Dr. Kendall Nygard

Dr. Limin Zhang

Approved:

| | |
|---|---|
| 4/14/2016 | Brian M. Slator |
| Date | Department Chair |

## ABSTRACT

Software inspection in the early phase of software development is proven to be an effective method to help developer to detect and fix defects in software requirement. It can improve the quality of software requirement, which affects the overall quality of the subsequent phases and hence, the final software product. In order to make inspection to be more effective, research focus on learning the factors that positively impact the performance of individual and team inspection is necessary. This paper presents a tool that can assist researchers to study the relationship between software inspectors' LS preferences and their performance in detecting defects during the inspection of software artifact. Several statistical techniques were employed in this tool, to create and sort different size of inspection teams based on dissimilarity of LS preferences of inspectors. This tool can be used to study correlations between individual inspector's LS strengths and their inspection team performance.

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Gursimran Singh Walia for his continued support throughout this paper. I appreciate his time, assistance and continuous guidance. I would also like to thank Dr. Kendall Nygard and Dr. Limin Zhang for being a part of my graduate supervisory committee. Thanks Anurag Goswami for his support and assistant for design of the tool. Special thanks to the faculty and staff of the Computer Science Department for their unconditional support throughout my master’s program.

## TABLE OF CONTENTS

## LIST OF FIGURES

# 1. INTRODUCTION

In today’s competitive world, software systems are widely used to automate vary tasks by organizations to improve the efficiency of production. Delivering a software in a timely and quality fashion is a key factor to the successfulness of software organizations [1]. In order to improve the quality of software artifacts, various approaches, including informal walkthroughs [2, 3], formal checklist based inspections [4, 5], prototyping [6] and testing [7], was used by software industries to help developers and managers uncovering and fixing defects in software artifacts. Defects can be introduced into software artifacts at various stages of software development. However if a defect is left undetected, it will penetrate and will become harder to find and fix at later stages of development [9]. It is proven in several studies that rework cost of detecting and fixing defects introduced in software artifacts in the early stage of Software Development Lifecycle (SDLC) is significantly lower than in later stages [8, 9, 10]. As a result, leading software organizations focus their attention on developing methods to aid developers and managers in finding and fixing faults at the early stages of software development [11, 12].

Requirements phase is the first and critical stage of software development, which involved many stockholders including both technical (developers, designers, testers) and non-technical (managers, end-users, sponsors). The major software artifact produced in this phase is Software Requirements Specification (SRS), where customer’s needs for developing the software are recorded using Natural Language (NL) in text base document. As a mean of communications among stakeholders, SRS helps in establishing a common understanding of problem and solution space for a software product. However, due to the inherit nature of NL, various problems, such as complexity, ambiguity, vagueness, and imprecision in information, can arise when requirements are written in NL [13-15]. It is widely recognized that the cost of fixing defects in released software can be as much as 80 times more than that of fixing them at the requirements stage. So it is essential to detect and fix faults in SRS in the requirements stage.

Due to the importance of correcting faults in SRS, many different approaches has been developed for detecting NL requirement faults, including NL to State transitions [16, 17], checklist based inspections [4], scenario based reading [18], ad hoc inspections [19]. Among these approaches, software inspections are widely accepted as the most effective technique. Software inspection process involves a group of skilled inspectors to review and uncover defects in a software artifact. Fagan inspection is a proven inspection process developed by Michael Fagan at IBM in the 1970s [2]. It generally includes the following steps: 1) appointing a moderator to organize inspection; 2) selecting inspectors to form an inspection panel; 3) having a kick-off meeting to introduce the objective of the inspection 4) individual review to find faults; 5) team meeting to consolidate faults; 6) moderator follow-up with author to repair. The output of this process is a list of faults presented in the artifact that can be fixed by the artifacts' author to avoid costly rework at the later stages [2, 20, 21].

Although every phase of the inspection process is important, Fagan [2] put more emphasis on an individual preparation phase rather than team meeting phase. The evidence shows that the performance of inspectors during the individual review significantly impacts overall inspection performance rather than the team meeting review [24]. To improve the effectiveness of individual inspection, research tried to understand the factors that could positively impact the performance of individual inspection. Intuitively, individual factors (e.g., educational background; the level of technical degree) are considered to be correlated to individual inspection effectiveness more possibly. A lot of studies focusing on these factors were conduct by researchers to evaluate the effect of educational background and level of technical degree of inspectors on their inspection effectiveness [22, 23]. Contrary to the expectations, empirical studies at major software organizations show that, inspectors with same technique and educational background varies significantly in inspection effectiveness. The result also shows that software engineers with a non-technical degree found significantly more requirement faults as compared to the technical degree holders [22]. As opposed to their technical expertise and level of education, it is possible that inspector's performance of uncovering defects in a software artifact is affected by some other psychological factors. So it is possible that the inspector's ability to find defects in a requirements document are affected by their individual strengths and preferences in the ways they comprehend and process information – i.e., their individual learning styles (LS).

Over the years, it has been proven by many cognitive psychology studies [25] that individuals have varying Learning Style (LS) preferences and strengths (i.e., individuals vary in the way they perceive and process information). For example, some people prefer to learn new things in small logical order; some are more comfortable learning in large jumps. Psychology research relating to LS also prove that individual can achieve a better and faster result in perceiving and process information when it is presented in their preferred LS [26, 27]. Many learning style models (such as The Myers-Briggs Type Indicator (MBTI), Kolb's Learning Style Model, and the Felder-Silverman Learning Style Model (FSLSM)) are developed and empirically evaluated by psychologists to assist the assessment of individual's LS. As software requirement is written in NL, inspectors should vary in the way they perceive and process information in SRS.

Although the concept of using cognitive-based approaches in software engineering domain is relatively new, it has been practiced in some researches. Previous research in software engineering has applied MBTI (a cognitive-based approach that measures psychological preference of individuals) into the process of creating heterogeneous inspection teams to maximize disparity between team members. They conclude that team with maximizing cognitive style dissimilarity will perform better than the inspection teams constructed with similar cognitive style [29]. There is another study that tries to use cognitive-based approach to migrate the communication problem among the stakeholders during distribute requirements elicitation. In their study, the researchers applied LS model of stakeholders into the process of selecting the requirements elicitation stakeholders. The results proved that learning preference of non-technical stakeholders also should be considered in order to get more suitable requirements elicitation method.

Based on above studies, it is highly possible that using inspector's Learning Styles (LS) to generate heterogeneous inspection teams can increase team performance by detecting more unique faults (i.e. less fault overlap) during the inspection. There is a need to analyze the impact of LS's of inspectors on team performance of inspection. So to simplify and smooth the analysis process, an easy to operate tool need to be developed.

This paper presents an easy operating GUI based tool to facilitate automatic generation of heterogeneous virtual inspection teams. By taking the LS preferences of the individual inspector, the tool can generate teams consisting of inspectors of dissimilar to similar LS's using statistical techniques. It can be used to analyze the impact of LS's of inspectors on team performance of inspection by following these steps: 1) creating virtual inspection teams by taking individual LS data for different team sizes; 2) sorting all virtual teams for each team size from most dissimilar to most similar in terms of the LS's of individual inspectors; 3) combining inspection data with virtual teams data to evaluate team performances by applying different cost models. In addition, Software managers can use these results to plan and manage inspections in their organizations.

# 2. BACKGROUND

Some background terms used in research approaches and tool will be described and explained. Section 2.1 introduced major steps of inspection. Section 2.2 inspection cost model which are used to create high performance inspection team. Section 2.3 described the concept of Learning Styles and Learning Styles Model used to measure individual's learning preference.

## 2.1. Inspection Process

Fagan inspection is a structured inspection process developed by Michael Fagan at IBM in the 1970s [2].  It is widely used and is empirically validated [2, 33, 34] for early detection and elimination of fault in software artifacts. Many variations [35, 36] of Fagan's original inspection concept were introduced based on different parts of the inspection process they emphasize on (e.g., variation that emphasis more on the individual preparation phase than the team meeting phase). Before inspections, a moderator will be appointed to organize the inspection and selects a team of inspectors. Next, a kick off meeting will be held to introduce inspectors some background and the purpose of the inspection. The inspection materials will be distributed among team members during the meeting. After the meeting, every inspector reads the document to detect and log faults in a fault form individually. Once individual inspection is completed, moderator will schedule another meeting for inspectors and author to discuss and verify the inspection result to create a master list of faults. After that, the moderator will send the fault list to the author and follow with author to fix these faults.

## 2.2. Inspection Cost Model

### 2.2.1. Inspection Cost Components

To evaluate the benefits of software inspection, inspection cost model [37] was developed to illustrate how much cost is saved if a fault is detected during inspections as compared to software testing. It has following components:

- $C_r$ – cost spent on an inspection (the sum of the total time taken to perform the inspection process in terms of man hours).
- $D_{total}$ – total number of faults present in the software product before the inspection.
- $D_r$ – number of unique faults detected during the inspection by all inspectors.
- $c_t$– average cost to detect a fault in testing.
- $C_t$ – testing cost: cost to detect remaining faults in testing if these fault wasn't detected during inspection. If a fault is left to detect in testing, the cost is usually much more expensive than the cost to detect it in inspection. The testing cost can be measured as the product of total number of faults remaining after inspection ($D_{total} – D_r$) and the average cost to detect a fault during testing ($c_t$). This is, $C_t= (D_{total} – D_r) * c_t$
- $\Delta C_t$ – testing cost saved during the testing if spending cost $C_r$ during inspection, which is calculated as the product of the total number of unique faults found during the inspection by all inspectors ($D_r$) and the average cost to detect a fault in testing ($c_t$). That is, $\Delta C_t = D_r * c_t$
- $C_{vt}$ – virtual testing cost, that is total testing cost if no inspections are performed. It is calculated as addition of testing cost ($C_t$) and the testing cost saved by inspection ($\Delta C_t$). That is, $C_{vt}= C_t+\Delta C_t$.

### 2.2.2. Kusumoto Cost Metric (Mk)

Kusumoto et al. [37] proposed a metric for evaluating the cost effectiveness of the inspection in terms of reduction of cost to detect and remove all defects from software product. Mk is a ratio of the saving costs to detect and remove all faults using inspections in a project ($\Delta C_t$-$C_r$) to the virtual testing cost if no inspection is executed ($C_t$+$\Delta C_t$), derived as:

$$Mk = \frac{(\Delta C_t - C_r)}{(C_t + \Delta C_t)}$$

The advantage of the model proposed by Kusumoto is that it normalizes the inspection saving cost by using the potential fault cost (ie. virtual testing cost). So it can be compared across different inspections and projects. This advantage makes Mk to be very appropriate for research purpose. Mk can also be used as a measurement of cost-effectiveness as it can be interpreted as the percentage of fault rework savings due to inspections. In this tool, Mk is used to evaluate the cost effectiveness of inspection teams generated and sorted based on the LS’s of the inspectors (i.e. dissimilar, similar and no preference).

## 2.3. Learning Styles

### 2.3.1. Inspection Cost Components

The concept of LS’s and the LS measurement instrument was firstly introduced by Kolb [38]. Over the years there are different variations of LS models [27, 38-43] developed by many educational psychologists. Among these LS models, the Felder Silverman Learning Style Model (FSLSM) is recognized as the most advanced and widely used model to capture most important LS preferences among individuals [25, 44, 45]. The FSLSM model used the instrument called Index of Learning Styles (ILS) to measure LS of an individual [44, 46], which classified individuals based their characteristic strength and preferences for the way they “perceive” and “process” information across four LS dimensions. Among these four dimensions, two dimensions (i.e., Sensing/Intuitive; and Visual/Verbal) are related to information perceiving and the other two dimensions (i.e. Active/Reflective and Sequential/Global) are related to information processing. Following is the description of four dimensions:

a. **Active/ Reflective**: **Active** people tend to understand information by trying it out. Furthermore, they prefer to learn by working in groups where they can discuss about the learned material; **Reflective** people prefer to think things through and understand things before acting. Regarding communication, they prefer to work alone or maybe in a small group together with one good friend.

b. **Sensing/Intuitive: Sensing** people prefer to deal with information that is concrete and practical. They are oriented towards details, facts, and figures and dislike complications and surprises. They like solving problems by well-established methods and proven procedure. Furthermore, sensing learners are considered to be more realistic and practical and like to relate the learned material to the real world; **Intuitive** people often prefer to take in information that is abstract, original, and oriented towards theory. They like innovation and dislike repetition.  Intuitive learners do not like work that involves a lot of memorization and routine calculations.

c. **Visual/ Verbal: Visual** people prefer visual presentations of material (such as pictures, diagrams, flow charts, time lines, films, and demonstrations). They prefer visually presented information; **Verbal** people get more out of words, and written and spoken explanations. They prefer verbally presented information

d. **Sequential/ Global: Sequential** people prefer linear thinking and learn something in small incremental steps. They tend to work with information in an organized and systematic way; **Global** people prefer to understand information in almost random manner without seeing connection. Their think usually appear in a scattered and disorganized way that is hard to understand, but they can offer have a creative and correct solution in the end.

# 3. RESEARCH APPROACHES

This section present several statistical techniques utilized by the software tool to generate virtual inspection using the LS's of inspectors, including Principal Component Analysis (PCA), Cluster Analysis (CA) and Discriminant Analysis (DA). PCA is used to eliminate any correlation of data by creating PC's based on LS data. CA uses the uncorrelated LS data generated by PCA to form clusters that include individuals with similar LS preferences. DA uses the results from PCA and CA to evaluate and improve the classification of CA, and to determine the dissimilarity of individuals LS preferences within one cluster.

## 3.1. Principal Component Analysis (PCA)

PCA is a statistical technique that is used to convert a set of observations of possibly correlated variables into set of values of uncorrelated variables called principal components (PCs) [47]. In FSLSM, each LS dimension is clarified into two categories (sensing/intuitive, visual/verbal, active/reflective and sequential/global) and they are negatively correlated. When a score in one category increases, the score in the other decreases. The dependency between two categories in each LS dimension may have effect on the result of grouping inspectors based similarity of LS preference in CA. So the software tool will apply PCA to remove dependency between two categories in each LS dimension. PCA transforms the original correlated LS variables into a new set of equal number of uncorrelated variables [47, 48]. Each PC is independent to other PC's and accounts for certain variance between the categories in each LS dimension; and between the dimensions, which reveals different properties of the original data.

The PCA starts to account maximum possible variance with the first PC and keep trying to account maximum possible variance that could not be explained by the last PC using the next PC until it can explain 100% variance of original data. Finally, it takes all possible number of

PC’s to explain 100% variance of original data. However, sometimes it is possible that the 100% variance coverage require less than total possible number of PC’s [49]. The end result of PCA (PC’s) is always listed in descending order with their respective variance. The total variation covered in the original data increases as the numbers of PCs are increasing. As the number of PC’s increases, the amount of variance described in original data decreases. After the last PC, the variance that is not covered in the original data is close to zero.

## 3.2. Cluster Analysis (CA)

CA is a statistical technique to form clusters (groups) with the objects that are relatively homogeneous within themselves and heterogeneous between other objects [47]. It is used in the tool to form group of individual inspectors based on their LS preferences, so inspectors will have high similarity of LS’s within one cluster and high dissimilarity of LS’s between different clusters [50].

There are broadly two types of clustering techniques - Hierarchical and None-Hierarchical (also called Partitional). Hierarchical clustering is a type of CA which create a hierarchical decomposition of the set of objects using some criterion. Hierarchical clustering could be future implemented in two approaches: the first one is a bottom up approach (also called Agglomerative). It starts with each observation in its own cluster, then find the best pair to merge into a new cluster. This process is repeated until all clusters are fused together. The second one is a top down approach (also called Divisive). It starts with all the observations in a single cluster and divide the cluster into two dissimilar groups. Then it will recursively operate on both sides until there are as many subgroups as observations. None-Hierarchical clustering is a type of clustering that construct various partitions and then evaluate and place each object into one of the clusters. The k-means clustering is non-hierarchical CA used in the tool to group objects. It is implemented by minimizing the sum of squares of distances between objects and their corresponding cluster centroid. In k-means [51], the algorithm firstly chooses k (k is defined by user) initial centroids at random. Then each object is assigned to the nearest centroid. After that, all the centroids are re-calculated and reset by using the mean distance of their associate objects. In last step, the algorithm will reassign objects to the new closest centroid. This process is repeated until there are no more changes in cluster [52].

## 3.3. Discriminant Analysis (DA)

DA is a statistical method used in the tool to sort individuals in a cluster based on their similarity to its cluster. By using DA, the tool is able to maximize the LS variations across different clusters, and minimize the LS variations within each cluster [47, 53].

While CA explained the dissimilarity among different clusters, there is a lack of assessment of dissimilarity of objects within the same cluster. DA is able to sort the teams ranging from most dissimilar LS to teams with most similar LS preferences and strengths. DA use Group Membership (GM) to measure the dissimilarities among individual LSs within the same cluster. The dissimilarities between each individual LS's within the same cluster could be evaluated by comparing the GM values of individuals. The higher GM value an individual LS's get, the higher similarity of it has. Therefore, DA delivers GM values to rank individuals in a cluster based on their similarity to the cluster.

DA also serves as a method to assess the adequacy of CA result (clusters information of each individual that is generated by CA) [54].  There is evidence from various literatures to show that the DA classification can help remove the misclassification that often plagues the CA output [53, 55]. To assess the adequacy of CA result, DA uses GM value of each individual and cluster combination (generated by CA). Then, DA considers the highest GM value of each individual and cluster combination; and assigns the individual into a cluster for which that individual has maximum GM value. In that way, DA classifies all the individuals into known clusters (that were generated by CA). Even there is very little chance of DA classification to be different from the classification result from CA, the CA classification is replaced with DA classification when the DA classification is different from CA classification.

# 4. APPLICATION OF RESEARCH TOOL

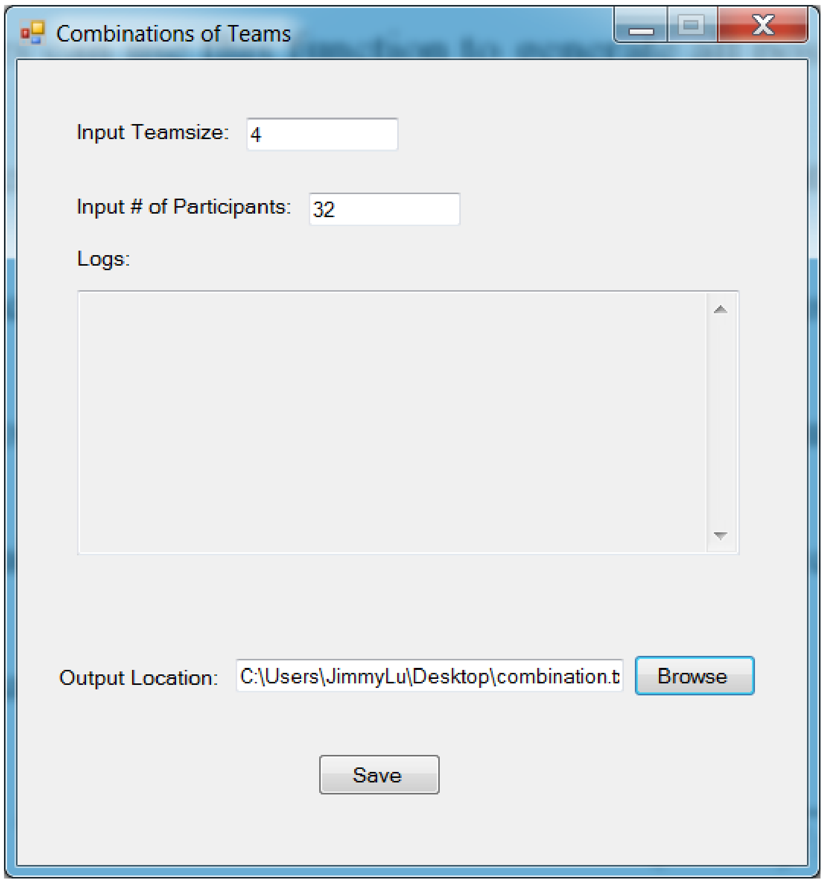
This section presents an automated tool that automates the formation of team development based on varying LS preferences, and maps it to the defect data of individual inspectors belonging to an inspection team to generate inspection cost data. Base on inspection data, the tool is able to generate inspection teams of equal strength.

## 4.1. Generate Combination of Teams

This function is to generate all possible combinations of teams for a given team size and a given number of participates, then user can save the result into a txt file. It lists all possible groups without repetition. Given ‘n’ individual inspectors, and an inspection team size of ‘r’, all possible combinations (teams) can be obtained using the equation

$$Number\ of\ combinations\ = \frac{n!}{(n-r)!\,r!}$$

For example, user wants to generate inspection teams of 4 inspectors from a pool of 32 inspectors and save the result into a txt file in desktop (as shown in Figure 1). After user click save button, the result of combination of teams will be saved into the output location. (as shown in Figure 2).

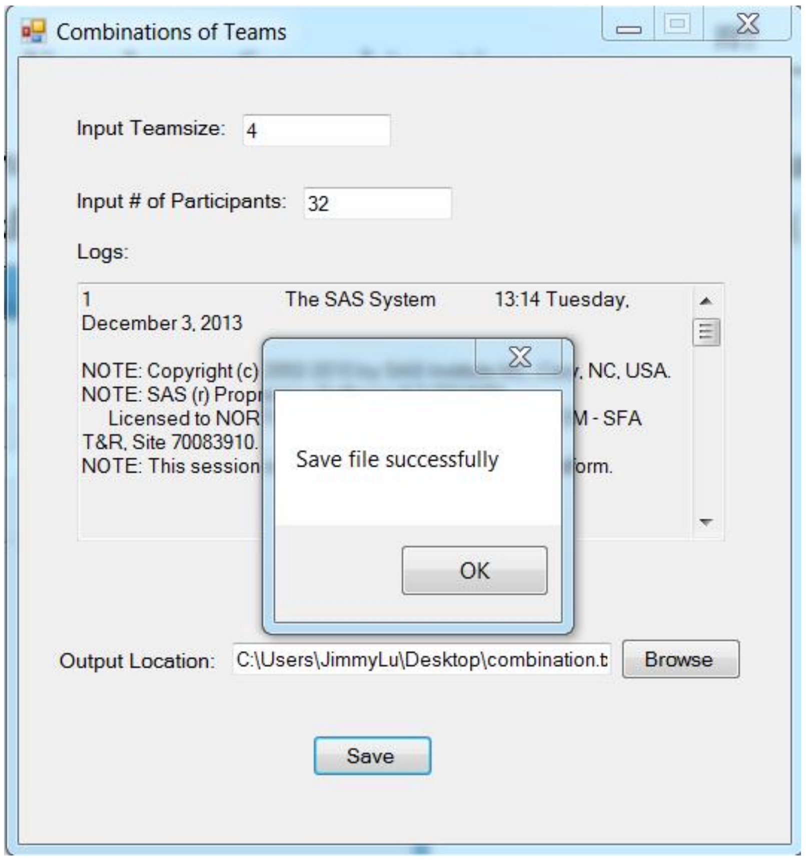


Input Teamsize: 4

Input # of Participants: 32

Output Location: C:\Users\JimmyLu\Desktop\combination.t

**Figure 1**. Team size and Number of Participants Entry



Input Teamsize: 4

Input # of Participants: 32

```
1                          The SAS System          13:14 Tuesday,
December 3, 2013

NOTE: Copyright (c)                                  , NC, USA.
NOTE: SAS (r) Propr
    Licensed to NOR                                  M - SFA
T&R, Site 70083910.
NOTE: This session                                   form.
```

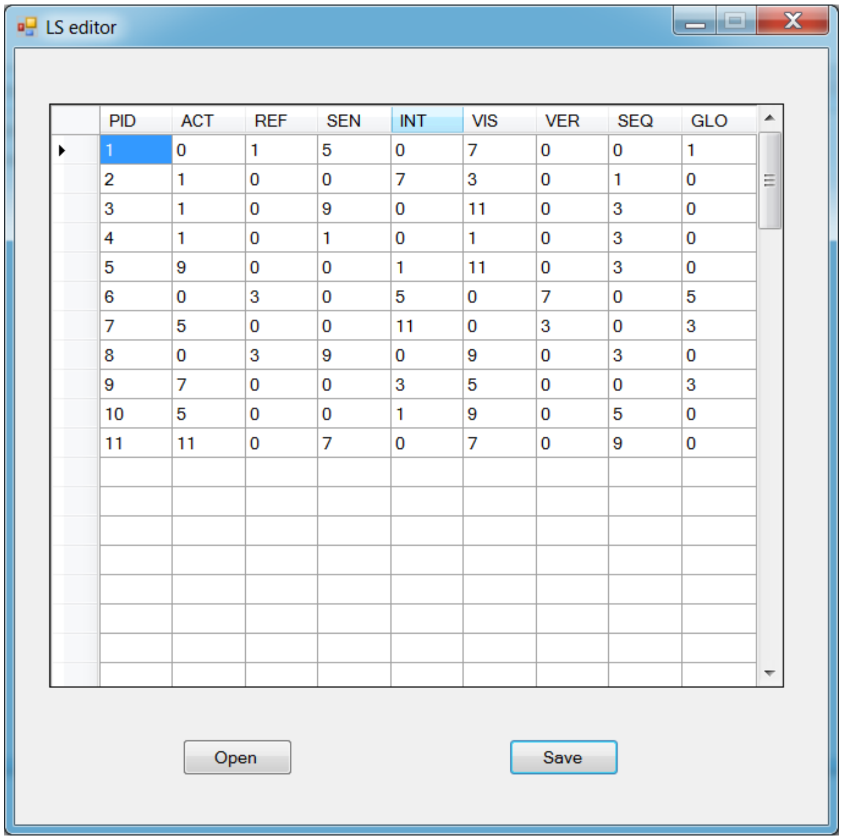
Save file successfully

Output Location: C:\Users\JimmyLu\Desktop\combination.t

**Figure 2**. Combination Generator

## 4.2. Create LS Data

This function allows user to enter learning style data into a table and save it into a txt file. Users need to enter all eight learning styles data along with inspector's ID (as shown in Figure 3). Also, user can edit a learning style data file by opening the txt file.



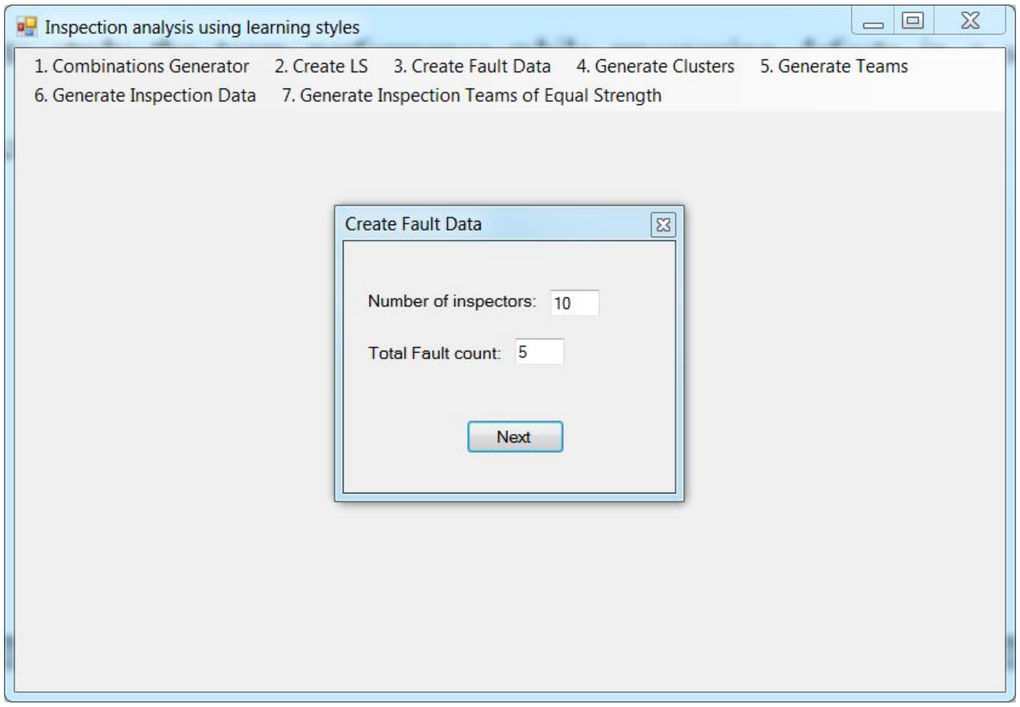
| PID | ACT | REF | SEN | INT | VIS | VER | SEQ | GLO |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 5 | 0 | 7 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 7 | 3 | 0 | 1 | 0 |
| 3 | 1 | 0 | 9 | 0 | 11 | 0 | 3 | 0 |
| 4 | 1 | 0 | 1 | 0 | 1 | 0 | 3 | 0 |
| 5 | 9 | 0 | 0 | 1 | 11 | 0 | 3 | 0 |
| 6 | 0 | 3 | 0 | 5 | 0 | 7 | 0 | 5 |
| 7 | 5 | 0 | 0 | 11 | 0 | 3 | 0 | 3 |
| 8 | 0 | 3 | 9 | 0 | 9 | 0 | 3 | 0 |
| 9 | 7 | 0 | 0 | 3 | 5 | 0 | 0 | 3 |
| 10 | 5 | 0 | 0 | 1 | 9 | 0 | 5 | 0 |
| 11 | 11 | 0 | 7 | 0 | 7 | 0 | 9 | 0 |

**Figure 3**. LS Data Entry

### 4.3. Create Fault Data Matrix

This function allows user to create a fault data matrix and save it into a txt file by using this function. Before generating the matrix, user has to enter the number of inspectors and total faults counts (as shown in Figure 4). In very first two columns, user can enter fault importance (either 1,2,3,4) and fault type (either G,AI,II,MI,MF,MP,ME,WS,O,EF) for each fault count (as shown in Figure 5). In the rest columns, user can enter the faults found of each inspector by replace '0' by '1' where fault is found in that fault count (as shown in Figure 6). Also, user needs to enter time each inspector used in inspection in last attribute of each column (as shown in Figure 7). After completed the fault data matrix, user can save it into a txt file. The fault data will be used to evaluate the total & redundant number of faults found by group of inspectors.

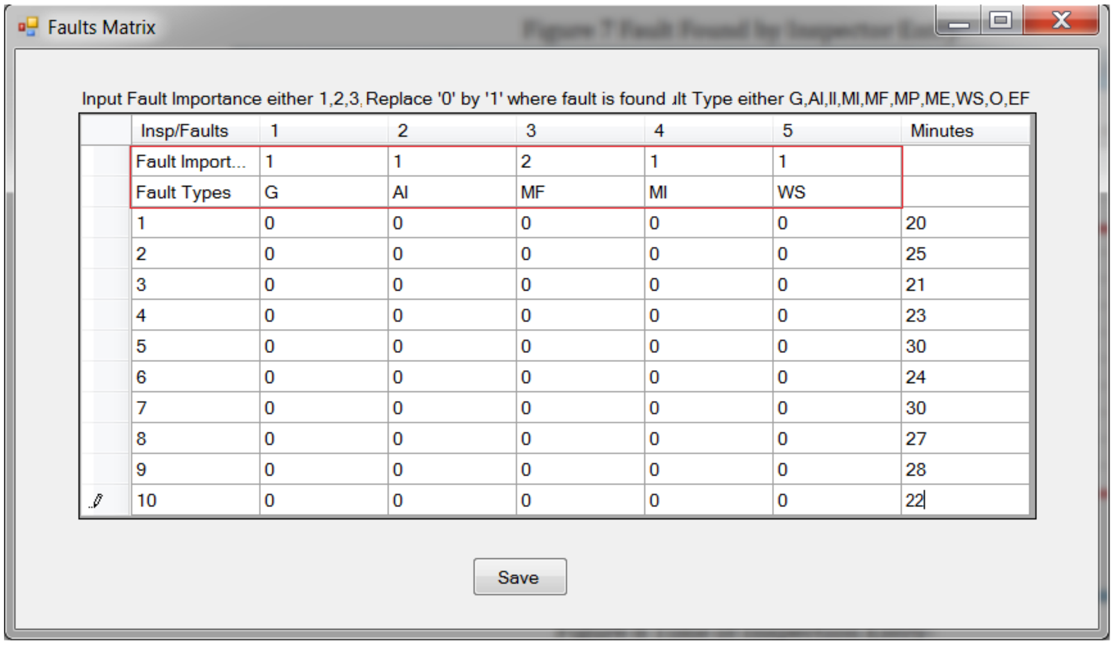
Researchers can fetch the advantage of faults data to study the team performance while uncovering defects in a software artifact.


Inspection analysis using learning styles
1. Combinations Generator
2. Create LS
3. Create Fault Data
4. Generate Clusters
5. Generate Teams
6. Generate Inspection Data
7. Generate Inspection Teams of Equal Strength
Create Fault Data
Next

Number of inspectors: 10

Total Fault count: 5

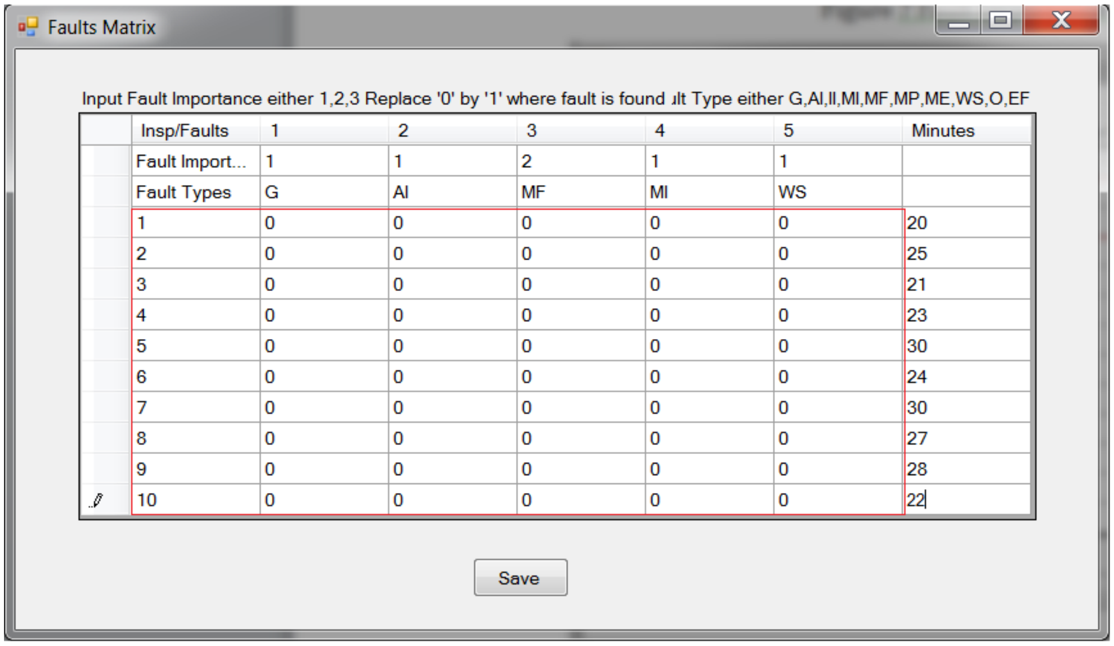
**Figure 4**. Number of Inspectors and Total Faults Counts Entry



Input Fault Importance either 1,2,3, Replace '0' by '1' where fault is found ılt Type either G,AI,II,MI,MF,MP,ME,WS,O,EF

| Insp/Faults | 1 | 2 | 3 | 4 | 5 | Minutes |
|---|---|---|---|---|---|---|
| Fault Import... | 1 | 1 | 2 | 1 | 1 | |
| Fault Types | G | AI | MF | MI | WS | |
| 1 | 0 | 0 | 0 | 0 | 0 | 20 |
| 2 | 0 | 0 | 0 | 0 | 0 | 25 |
| 3 | 0 | 0 | 0 | 0 | 0 | 21 |
| 4 | 0 | 0 | 0 | 0 | 0 | 23 |
| 5 | 0 | 0 | 0 | 0 | 0 | 30 |
| 6 | 0 | 0 | 0 | 0 | 0 | 24 |
| 7 | 0 | 0 | 0 | 0 | 0 | 30 |
| 8 | 0 | 0 | 0 | 0 | 0 | 27 |
| 9 | 0 | 0 | 0 | 0 | 0 | 28 |
| 10 | 0 | 0 | 0 | 0 | 0 | 22 |

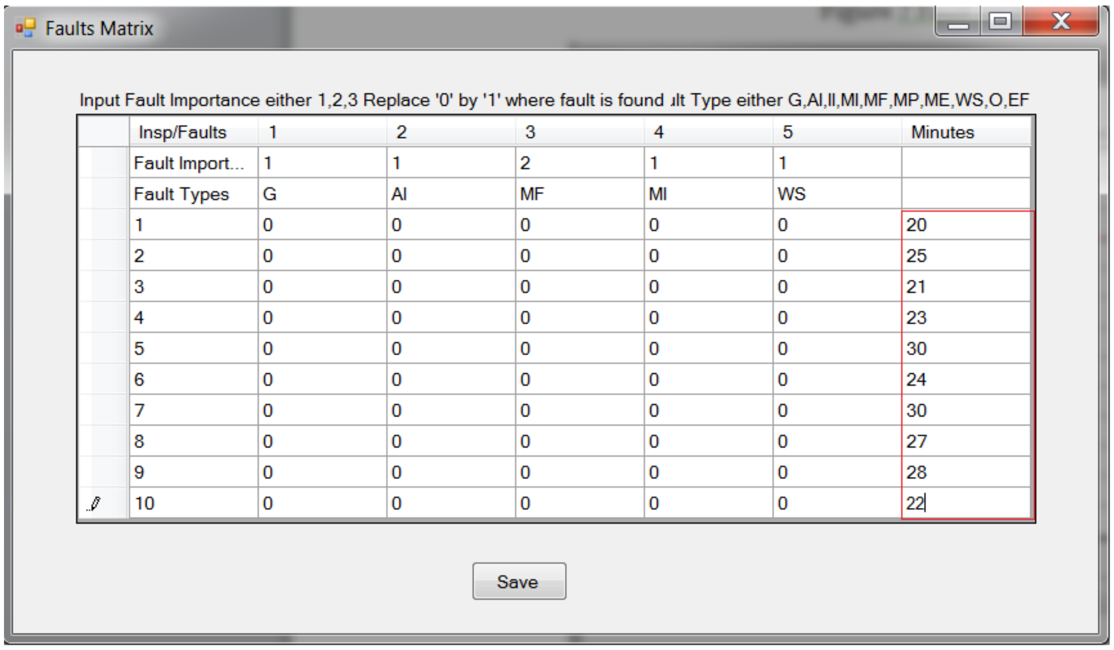
**Figure 5**. Fault Importance and Fault Types Entry



Input Fault Importance either 1,2,3 Replace '0' by '1' where fault is found ılt Type either G,AI,II,MI,MF,MP,ME,WS,O,EF

| Insp/Faults | 1 | 2 | 3 | 4 | 5 | Minutes |
|---|---|---|---|---|---|---|
| Fault Import... | 1 | 1 | 2 | 1 | 1 | |
| Fault Types | G | AI | MF | MI | WS | |
| 1 | 0 | 0 | 0 | 0 | 0 | 20 |
| 2 | 0 | 0 | 0 | 0 | 0 | 25 |
| 3 | 0 | 0 | 0 | 0 | 0 | 21 |
| 4 | 0 | 0 | 0 | 0 | 0 | 23 |
| 5 | 0 | 0 | 0 | 0 | 0 | 30 |
| 6 | 0 | 0 | 0 | 0 | 0 | 24 |
| 7 | 0 | 0 | 0 | 0 | 0 | 30 |
| 8 | 0 | 0 | 0 | 0 | 0 | 27 |
| 9 | 0 | 0 | 0 | 0 | 0 | 28 |
| 10 | 0 | 0 | 0 | 0 | 0 | 22 |

**Figure 6**. Fault Found by Inspector Entry



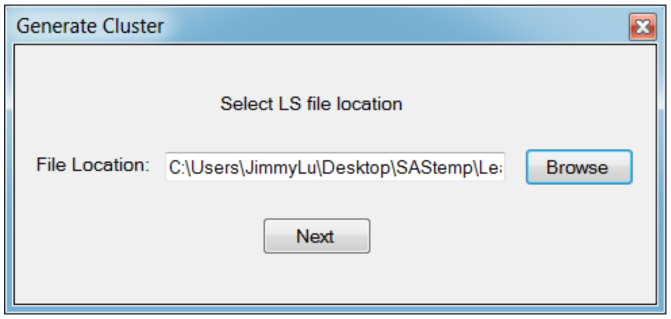
Input Fault Importance either 1,2,3 Replace '0' by '1' where fault is found ılt Type either G,AI,II,MI,MF,MP,ME,WS,O,EF

| Insp/Faults | 1 | 2 | 3 | 4 | 5 | Minutes |
|---|---|---|---|---|---|---|
| Fault Import... | 1 | 1 | 2 | 1 | 1 | |
| Fault Types | G | AI | MF | MI | WS | |
| 1 | 0 | 0 | 0 | 0 | 0 | 20 |
| 2 | 0 | 0 | 0 | 0 | 0 | 25 |
| 3 | 0 | 0 | 0 | 0 | 0 | 21 |
| 4 | 0 | 0 | 0 | 0 | 0 | 23 |
| 5 | 0 | 0 | 0 | 0 | 0 | 30 |
| 6 | 0 | 0 | 0 | 0 | 0 | 24 |
| 7 | 0 | 0 | 0 | 0 | 0 | 30 |
| 8 | 0 | 0 | 0 | 0 | 0 | 27 |
| 9 | 0 | 0 | 0 | 0 | 0 | 28 |
| 10 | 0 | 0 | 0 | 0 | 0 | 22 |

**Figure 7**. Inspection Time Entry

### 4.4. Create Cluster

This function allows user to create cluster for each inspector based on LS preferences. To create cluster, user needs to select a Learning Style file as an input file (as shown in Figure 8). Then user can select LS dimensions in Cluster Formation (as shown in Figure 9), selected dimensions will be used to classify each inspector into different clusters. By clicking the ‘Continue’ button, the application will show the result in a new window. Cluster Result window (as shown in Figure 10) shows a list of inspector identifiers and their corresponding clusters. Each cluster stands for one LS combination, which is displayed under Cluster Lable. User can save the result into a txt file by clicking ‘Browser’ button to select an output location and then clicking ‘Save’ button to save the file.

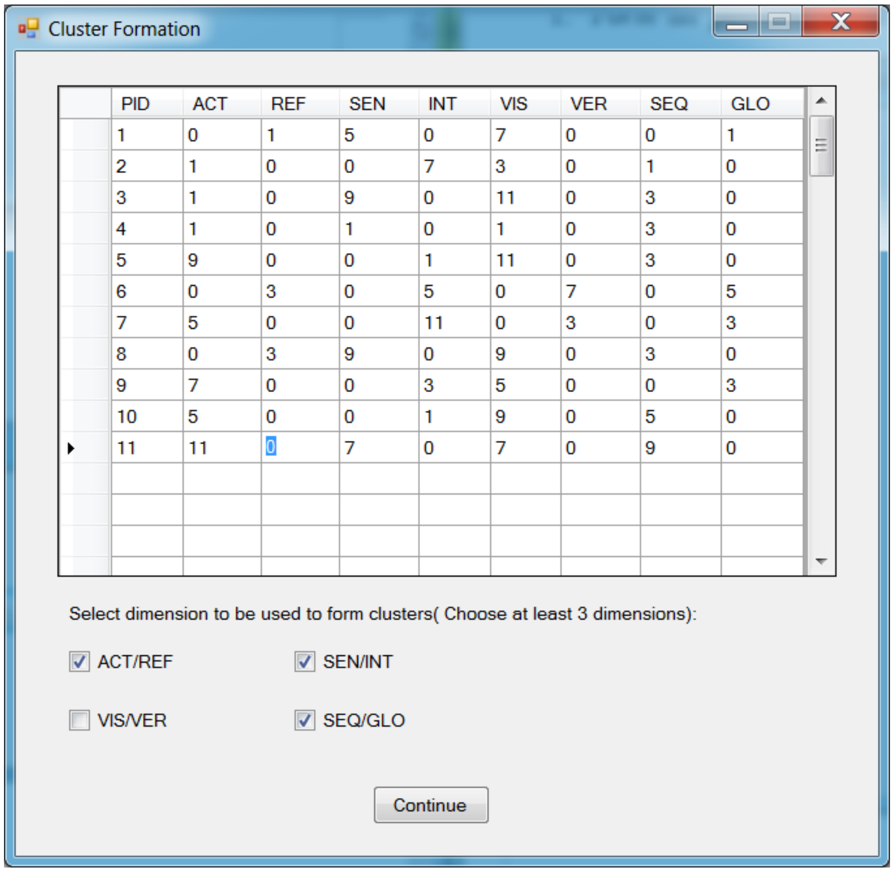


Generate Cluster

Select LS file location

File Location: C:\Users\JimmyLu\Desktop\SAStemp\Le:

**Figure 8**. LS File Location Selection for Cluster Generation



Cluster Formation

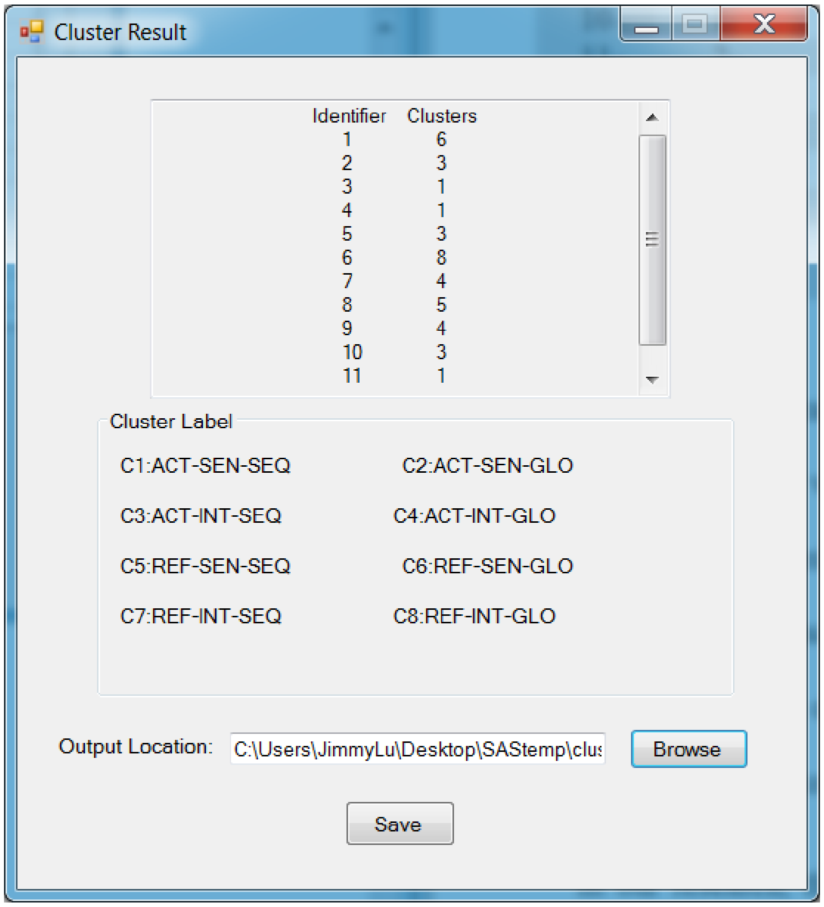
| PID | ACT | REF | SEN | INT | VIS | VER | SEQ | GLO |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 5 | 0 | 7 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 7 | 3 | 0 | 1 | 0 |
| 3 | 1 | 0 | 9 | 0 | 11 | 0 | 3 | 0 |
| 4 | 1 | 0 | 1 | 0 | 1 | 0 | 3 | 0 |
| 5 | 9 | 0 | 0 | 1 | 11 | 0 | 3 | 0 |
| 6 | 0 | 3 | 0 | 5 | 0 | 7 | 0 | 5 |
| 7 | 5 | 0 | 0 | 11 | 0 | 3 | 0 | 3 |
| 8 | 0 | 3 | 9 | 0 | 9 | 0 | 3 | 0 |
| 9 | 7 | 0 | 0 | 3 | 5 | 0 | 0 | 3 |
| 10 | 5 | 0 | 0 | 1 | 9 | 0 | 5 | 0 |
| 11 | 11 | 0 | 7 | 0 | 7 | 0 | 9 | 0 |

Select dimension to be used to form clusters( Choose at least 3 dimensions):

[✓] ACT/REF [✓] SEN/INT

[ ] VIS/VER [✓] SEQ/GLO

**Figure 9**. Cluster Formation



| Identifier | Clusters |
|---|---|
| 1 | 6 |
| 2 | 3 |
| 3 | 1 |
| 4 | 1 |
| 5 | 3 |
| 6 | 8 |
| 7 | 4 |
| 8 | 5 |
| 9 | 4 |
| 10 | 3 |
| 11 | 1 |

Cluster Label

C1:ACT-SEN-SEQ C2:ACT-SEN-GLO

C3:ACT-INT-SEQ C4:ACT-INT-GLO

C5:REF-SEN-SEQ C6:REF-SEN-GLO

C7:REF-INT-SEQ C8:REF-INT-GLO

Output Location: C:\Users\JimmyLu\Desktop\SAStemp\clus

**Figure 10**. Cluster Result

### 4.5. Generate Inspection Teams

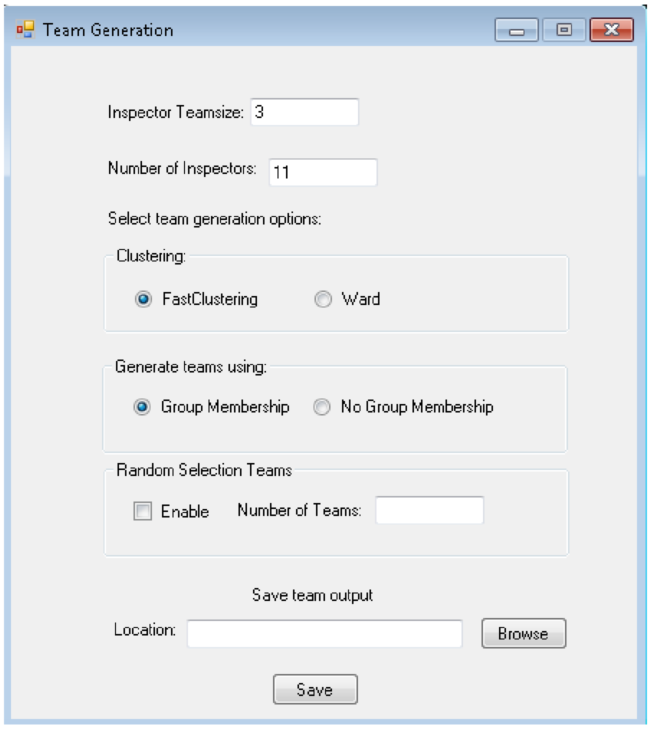
This function is to generate an inspection teams with a given size. Firstly, user needs to select a Learning Style file as an input file. After that, user clicks ‘Continue’ button to go to Team Generation window (as shown in Figure 15). To generate teams, user has to enter the size of inspection team and number of inspectors same to the number of inspector in LS file. Then user can select clustering method either in Fast Clustering or Ward Clustering and generate team with Group Membership or NO Group Membership. After selecting output location by clicking ‘Browse’ button, user can click ‘Save’ button to save the team data into a txt file.

Ward Clustering: The Ward Clustering performs hierarchical clustering of observations by using agglomerative methods applied to coordinate data or distance data.

Fast Clustering: The Fast Clustering is an effective clustering method using K- mean algorithm. In Fast Clustering, a set of points called cluster seeds is selected as a first guess of the means of the clusters. Each observation is reassigned to the nearest seed to form temporary clusters. The seeds are then replaced by the means of the temporary clusters, and the process is repeated until no further changes occur in the clusters.  The advantage of Fast Clustering is that it can handle large amount of data in an efficient way. It is recommended to use for team generation in this tool.

Group Membership or NO Group Membership: By selecting Group Membership, the tool will apply Discriminant Analysis to the team generation. So the virtual inspection will be sort ranging from most dissimilar LS to teams with most similar LS preferences.

Random Selection Team: By enable this function, the tool is able to generate a number of virtual inspection team specified by user randomly.  It can save user a lot of time rather than generating all teams if they only need a small number of teams for research and other purposes.



Team Generation

Inspector Teamsize: 3

Number of Inspectors: 11

Select team generation options:

Clustering: [✓ FastClustering] Ward

Generate teams using: [✓ Group Membership] No Group Membership

Random Selection Teams

Enable ☐ Number of Teams:

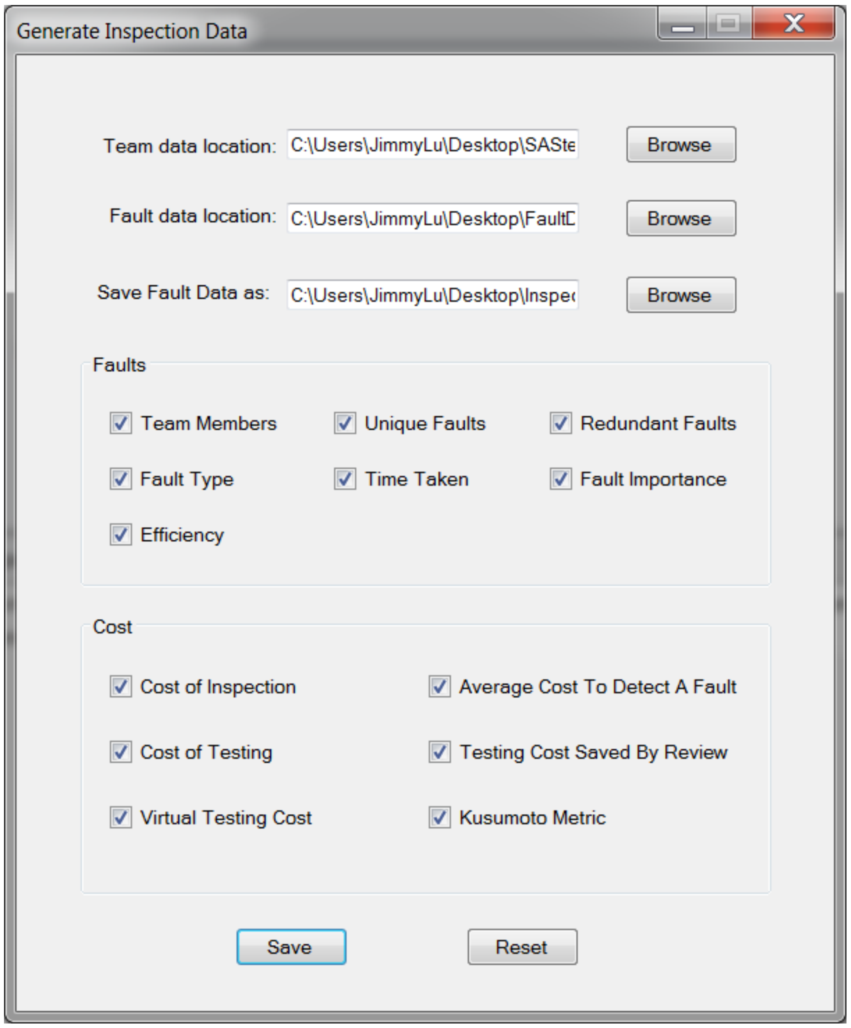
Save team output

Location:

**Figure 11**. Team Generation

### 4.6. Generate Inspection Cost Data

This function allows user to generate inspection cost data. To generate data, user needs to select team data file (created in the function 5) and fault data file (created in the function 3) as input files. After that, user can select location of output and check what kind of inspection data will be included in the output files (as shown in Figure 12). By clicking ‘Save’ button, a txt file of inspection cost data will be generated in the location user selected (as shown in Figure 13).


Generate Inspection Data

Team data location: C:\Users\JimmyLu\Desktop\SASte

Fault data location: C:\Users\JimmyLu\Desktop\FaultD

Save Fault Data as: C:\Users\JimmyLu\Desktop\Inspec

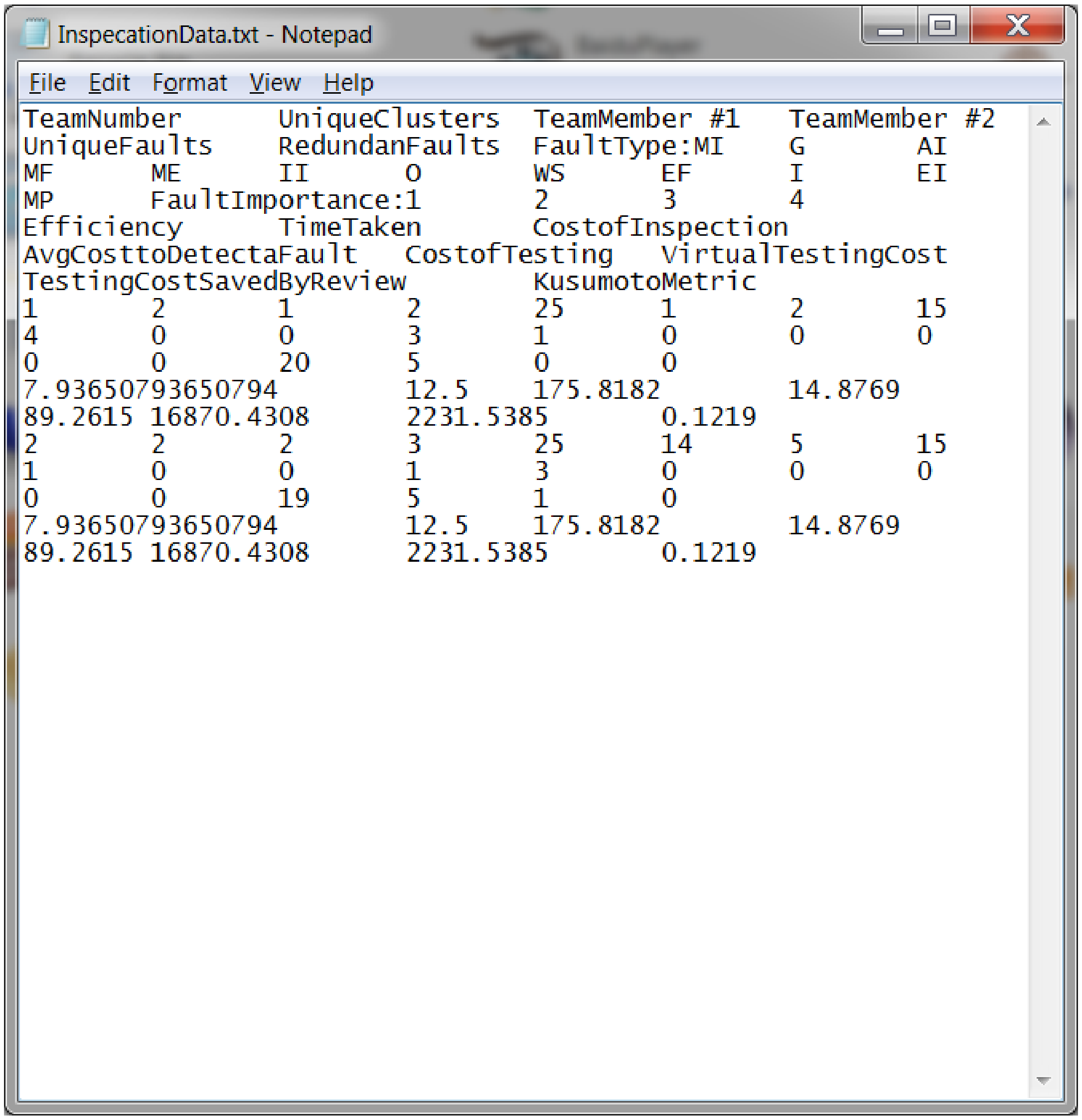
Faults

- [✓] Team Members
- [✓] Unique Faults
- [✓] Redundant Faults
- [✓] Fault Type
- [✓] Time Taken
- [✓] Fault Importance
- [✓] Efficiency

Cost

- [✓] Cost of Inspection
- [✓] Average Cost To Detect A Fault
- [✓] Cost of Testing
- [✓] Testing Cost Saved By Review
- [✓] Virtual Testing Cost
- [✓] Kusumoto Metric

**Figure 12**. Inspection Cost Data Generation



```
TeamNumber       UniqueClusters   TeamMember #1    TeamMember #2
UniqueFaults     RedundanFaults   FaultType:MI     G        AI
MF       ME      II       O       WS       EF      I        EI
MP       FaultImportance:1        2        3       4
Efficiency       TimeTaken        CostofInspection
AvgCosttoDetectaFault    CostofTesting    VirtualTestingCost
TestingCostSavedByReview          KusumotoMetric
1        2       1        2       25       1       2        15
4        0       0        3       1        0       0        0
0        0       20       5       0        0
7.93650793650794         12.5     175.8182         14.8769
89.2615  16870.4308      2231.5385         0.1219
2        2       2        3       25       14      5        15
1        0       0        1       3        0       0        0
0        0       19       5       1        0
7.93650793650794         12.5     175.8182         14.8769
89.2615  16870.4308      2231.5385         0.1219
```

**Figure 13**. Inspection Cost Data Text File

### 4.6.1. Fault Data

Fault data include Team member, Unique Faults, Redundant Faults, Fault Type, Fault Importance, Time Taken and Efficiency. Each option is described as below:

- Team member: By selecting ‘Team member’, inspectors IDs which are in the same virtual team will be included in the inspection data file.

- Unique Faults: By selecting ‘Unique Faults’, the number of total unique defeats found by each virtual team will be calculated and included in the inspection data file.
- Redundant Faults: By selecting ‘Redundant Faults’, the number of redundant defeats found by each virtual team will be calculated and included in the inspection data file.
- Fault Type: By selecting ‘Fault Type’, the number of defeats of each fault type found by each virtual team will be calculated and included in the inspection data file.
- Fault Importance: By selecting ‘Fault Importance’, the number of defeats in each fault importance level found by each virtual team will be calculated and included in the inspection data file.
- Time Taken: By selecting ‘Time Taken’, average time taken by each inspector in a virtual team will be calculated and included in the inspection data file.
- Efficiency: By selecting ‘Efficiency’, the efficiency of each virtual team will be calculated and included in the inspection data file. Given ‘m’ total unique faults found by the team, and ‘t’ total time taken in minutes by the team, the Efficiency will be calculated using the equation:

$$Efficiency = \mathrm{m}\frac{t}{60}$$

### 4.6.2. Cost Data

By using the inspection cost model mentioned in Section 3, inspection cost data can be generated by selecting following options: Cost of Inspection, Average Cost to Detect a Fault, Cost of Testing, Testing Cost Saved by Review, Virtual Testing Cost, Kusumoto Metric. Each option is described below:

- Cost of Inspection: By selecting ‘Cost of Inspection’, average time spent by each inspector in inspection will be calculated and included in the cost data file. Given ‘t’ total time spent by all inspector in inspection, and ‘n’ number of all inspector, the Cost of Inspection will be calculated using following equation:

$$\mathrm{C_r} = \frac{t}{n}$$

- Average Cost to Detect a Fault: By selecting ‘Average Cost to Detect a Fault’, average time spent to detect a fault in testing will be calculated and included in the cost data file. Given ‘t’ total time spent by all inspector in inspection, and ‘$\mathrm{D_r}$’ number of unique faults detected found during the inspection by all inspectors, the result will be calculated using the equation:

$$\mathrm{c_t} = \frac{\mathrm{t}}{\mathrm{D_r}} * 6$$

- Cost of Testing: By selecting ‘Cost of Testing’, cost to detect remaining faults in testing will be calculated and included in the inspection data file. Given ‘$\mathrm{D_{total}}$’ total number of faults present in the software product before the inspection, Cost of Testing is calculated by using the equation:

$$\mathrm{C_t} = (\mathrm{D_{total}} - \mathrm{D_r}) * \mathrm{c_t}$$

- Testing Cost Saved by Review: By selecting ‘Testing Cost Saved by Review’, testing cost saved by detecting faults in review will be calculated and included in the cost data file. Testing Cost Saved by Review is calculated by using the equation:

$$\Delta\mathrm{C_t} = \mathrm{D_r} * \mathrm{c_t}$$

- Virtual Testing Cost: By selecting ‘Virtual Testing Cost’, virtual testing cost will be calculated and included in the cost data file. Virtual testing cost is the cost to detect all unique faults in testing, if no inspection is executed. It is calculated by using the equation:

$$\mathrm{C_{vt}} = \mathrm{C_t} + \Delta\mathrm{C_t}$$

- Kusumoto Metric: By selecting ‘Kusumoto Metric’, Mk will be calculated and included in the inspection data file. Mk server as a measurement of the cost effectiveness of inspection. It is calculated by using the equation:

$$Mk = \frac{(\Delta \mathrm{C_t} - \mathrm{C_r})}{(\mathrm{C_t} + \Delta \mathrm{C_t})}$$

# 5. CONCLUSION AND FUTURE IMPROVEMENTS

This paper presents a software tool that help researchers to study the impact of inspectors' LS preferences on the inspection performance during the software artifact inspection. To achieve this goal, the tool helps to sort inspection teams with inspectors ranging from most similar to most dissimilar LS. Then, this tool allows researchers to map defect data to each inspector and calculate unique number of faults found by inspection team of varying sizes. After that, researchers can evaluate the inspection performance of teams with varying size and dissimilarity by comparing the amount of unique fault found by each team. In addition, this tool is able to provide data to facilitate project managers to select the software inspectors with larger potential of uncovering defects. Furthermore, this tool could be used to create diverse software development teams which could improve collaboration and overall quality of development.

In the future, this software tool is anticipate to be running on client- server basis. So user can just download and run light weight client application without installing heavy SAS engine. Also, generation of inspection teams (of particular size from given number of inspectors) of equal strength using their LS as an input is also a required future work for this tool.

## REFERENCES

1. Nalbant, S., & Nalbant, S. (2012). An evaluation of the reinspection decision policies for software code inspections. Retrieved April 11, 2016, from http://etd.lib.metu.edu.tr/upload/12605827/index.pdf
2. Fagan, M. E. (1986). *Advances in software inspections. Software Engineering IEEE Transactions on, 12*(7), 744-751.
3. Pressman, R. S. (1982). *Software engineering: A practitioner's approach*. New York: McGraw-Hill.
4. Parnas, D., & Lawford, M. (2003). The role of inspection in software quality assurance. *IEEE Transactions on Software Engineering IIEEE Trans. Software Eng., 29*(8), 674-676.
5. Laitenberger, O. (2002). A Survey Of Software Inspection Technologies. *Handbook of Software Engineering and Knowledge Engineering Volume II: Emerging Technologies In 2 Volumes*, 517-555. Retrieved April 11, 2016, from http://programmingresearch.com/content/misc/a-survey-of-sw-inspection-technologies-Laitenberger.pdf
6. Subramanian, G. H., Jiang, J. J., & Klein, G. (2007). Software quality and IS project performance improvements from software development process maturity and IS implementation strategies. *Journal of Systems and Software*, *80*(4), 616-627.
7. Tian, J. (2005). *Software quality engineering: Testing, quality assurance, and quantifiable improvement*. Hoboken, NJ: Wiley.

8. Freimut, B., Briand, L., & Vollei, F. (2005). Determining inspection cost-effectiveness by combining project data and expert opinion. *IEEE Transactions on Software Engineering IIEEE Trans. Software Eng., 31*(12), 1074-1092.
9. Briand, L., Freimut, B., & Vollei, F. (n.d.). Assessing the cost-effectiveness of inspections by combining project data and expert opinion. *Proceedings 11th International Symposium on Software Reliability Engineering. ISSRE 2000.*
10. Perry, W. E. (2006). *Effective Methods for Software Testing: Includes Complete Guidelines, Checklists, and Templates*. New York: Wiley.
11. Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., & Wong, M. (1992). Orthogonal defect classification-a concept for in-process measurements. *IEEE Transactions on Software Engineering IIEEE Trans. Software Eng., 18*(11), 943-956.
12. Leszak, M., Perry, D. E., & Stoll, D. (2000). A case study in root cause defect analysis. *Proceedings of the 22nd International Conference on Software Engineering - ICSE '00.*
13. Berry, D. M., & Kamsties, E. (2004). *Ambiguity in Requirements Specification. Perspectives on Software Requirements.* Springer US.
14. Berry, D. M. (2007). *Ambiguity in Natural Language Requirements Documents. Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs.* Springer Berlin Heidelberg.
15. Fabbrini, F., Fusani, M., Gervasi, V., Gnesi, S., & Ruggieri, S. (2000). Achieving quality in natural language requirements. *Proceedings of the Eleventh International Software Quality Week S Francisco Ca Software Research Institute.*

16. Aceituna, D., Do, H., Walia, G. S., & Lee, S. W. (2011). Evaluating the use of model-based requirements verification method: A feasibility study. Empirical Requirements Engineering (EmpiRE), *2011 First International Workshop on* (pp.13 - 20).

17. Aceituna, D., Walia, G., Do, H., & Lee, S. W. (2013). Model-based requirements verification method: conclusions from two controlled experiments. *Information & Software Technology, 56*(3), 321-334.

18. Shull, F., Rus, I., & Basili, V. (2000). How perspective-based reading can improve requirements inspections. *Computer, 33(7)(7),* 73-79.

19. Porter, A. A., Votta, L. G., & Basili, V. R. (1995). Comparing detection methods for software requirements inspections: a replicated experiment. *IEEE Transactions on Software Engineering, 21*(6), 563-575.

20. Fagan, M. E. (1999). Design and code inspections to reduce errors in program development. *IBM Systems Journal, 38*(2.3), 258-287.

21. Ackerman, A. F., Buchwald, L. S., & Lewski, F. H. (1989). Software inspections: an effective verification process. *Software IEEE, 6*(3), 31-36.

22. Carver, J. (2004). The impact of background and experience on software inspections. *Empirical Software Engineering, 9*(3), 259-262.

23. Carver, J. C., Nagappan, N., & Page, A. (2009). The impact of educational background on the effectiveness of requirements inspections: an empirical study. *IEEE Transactions on Software Engineering, 34*(6), 800-812.

24. Porter, A. A., Siy, H., Mockus, A., & Votta, L. G. (2001). Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering & Methodology, 7,* 41--79.

25. Felder, R. M., & Silverman, L. K. (1988). Learning and teaching styles in engineering. *Journal of Engineering Education, 78*(7), 674-681.
26. Allert, J. (2004). Learning Style and Factors Contributing to Success in an Introductory Computer Science Course. *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on* (pp.385-389). IEEE.
27. Charkins, R. J., O'Toole, D. M., & Wetzel, J. N. (2014). Linking teacher and student learning styles with student achievement and attitudes. *Journal of Economic Education, 16*(16), 111-120.
28. Rutherfoord, R. H. (2001). Using personality inventories to help form teams for software engineering class projects. *ACM Sigcse Bulletin, 33*(3), 73-76.
29. Miller, J., & Yin, Z. (2004). A cognitive-based mechanism for constructing software inspection teams. *IEEE Transactions on Software Engineering, 30*(30), 811-825.
30. Myers, I. B., Mccaulley, M. H., & Most, R. (1998). *Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. Consulting Psychologists Press.
31. Montgomery, S. M. (1995). Addressing diverse learning styles through the use of multimedia. *Frontiers in Education Conference, 1995. Proceedings* (Vol.1, pp.3a2.13-3a2.21 vol.1).
32. Aranda, G. N., Vizcaino, A., Cechich, A., & Piattini, M. (2005). A cognitive-based approach to improve distributed requirements elicitation processes. *Cognitive Informatics, 2005. (ICCI 2005). Fourth IEEE Conference on* (pp.322-330). IEEE.
33. Doolan, E. P. (1992). Experience with fagan's inspection method. *Software Practice & Experience, 22*(2), 173-182.

34. Russell, G. W. (1991). Experience with inspection in ultralarge-scale development. *IEEE Software, 8*(1), 25-31.

35. Martin, J., & Tsai, W. T. (1990). N-fold inspection: a requirements analysis technique. *Communications of the ACM, 33*(2), 225-232.

36. Parnas, D. L., & Weiss, D. M. (2001). *Active design reviews: Principles and practices. Software fundamentals.* Addison-Wesley Longman Publishing Co. Inc.

37. Kusumoto, S., Matsumoto, K., Kikuno, T., & Torii, K. (1992). A new metric for cost-effectiveness of software reviews. *Ieice Transactions on Information & Systems,* (5), 674-680.

38. Kolb, David. (1984). *Experiential Learning: Experience As the Source of Learning and Development.*

39. Kane, M. (1984). Cognitive Styles of Thinking and Learning. Part One. *Academic therapy, 19*(5), 527-536.

40. Peggy Friedman, & Robert Alley. (2010). Learning/teaching styles: applying the principles. *Theory Into Practice, 23*(1), 77-81.

41. Mccarthy, B. (1987). The 4mat system: teaching to learning styles with right/left mode techniques. *Excel*.

42. Myers, B. I., Mccaulley, M., Quenk, N., & Hammer. (2010). *A guide to the development and use of the myers-briggs type indicator, 3rd edn, Consulting Psychologist*.

43. Busato, V. V., Prins, F. J., Elshout, J. J., & Hamaker, C. (1998). The relation between learning styles, the big five personality traits and achievement motivation in higher education. *Personality & Individual Differences, 26*(98), 129–140.

44. Felder, R., & Spurlin, J. (2005). Applications, reliability, and validity of index of learning styles. *Int Journal of Engineering Education, 21*(1), 103-112.

45. Felder, R.M. (2010). Are learning styles invalid?(Hint: No!). *On-Course Newsletter,1-7.*

46. Felder, R. M., & Soloman, B. A. (1999). Index of learning styles. *Raleigh*.

47. Anderson, T.W. (1958). *An introduction to multivariate statistical analysis*. New York: Wiley.

48. Torbick, N., & Becker, B. (2009). Evaluating principal components analysis for identifying optimal bands using wetland hyperspectral measurements from the Great Lakes, USA. *Remote Sensing, 1*(3), 408-417.

49. Jolliffe, I. T. (2010). Principal component analysis. *Springer Berlin, 87*(100), 41-64.

50. Steinbach, M., Ertöz, L., & Kumar, V. (2004). *The Challenges of Clustering High Dimensional Data. New Directions in Statistical Physics.* Springer Berlin Heidelberg.

51. Hartigan, J. A., & Wong, M. A. (1979). Algorithm as 136: a k-means clustering algorithm. *Applied Statistics, 28*(1), 100-108.

52. Mackay, D. J. C. (2003). *InformationTheory, Inference, and Learning Algorithms*.

53. Tatsuoka, M. M., & Tiedeman, D. V. (1954). Chapter IV: discriminant analysis. *Review of Educational Research, 24*(5), 402-420.

54. Galbraith, J. K., & Lu, J. (1999). Cluster and discriminant analysis on time-series as a research tool. *Ssrn Electronic Journal*.

55. Klecka, W. R. (2005). *Discriminant Analysis.* SPSS: Statistical Package for the Social Sciences.

56. Mandala, N. R., Walia, G. S., Carver, J. C., & Nagappan, N. (2012). Application of Kusumoto cost-metric to evaluate the cost effectiveness of software inspections. *ACM-*

*IEEE International Symposium on Empirical Software Engineering and Measurement* (Vol.7304, pp.221-230). IEEE Computer Society.