Detecting Software Usability Deficiencies Through Pinpoint Analysis

Dan E. Tamir, Divya K. V. Dasari
Oleg V. Komogortsev, Gregory R. LaKomski
Department of Computer Science
Texas State University
San Marcos, Texas USA
{dt19, dd1290, ok11, gl1082}@txstate.edu

Carl J. Mueller
Department of Computer Information Systems
Texas A&M University Central Texas
Killeen, Texas, USA
muellercj@ct.tamus.edu

Abstract— **The effort-based model of usability is used for evaluating user interface (UI), development of usable software, and pinpointing software usability defects. In this context, the term pinpoint analysis refers to identifying and locating software usability deficiencies and correlating these deficiencies with the UI software code. For example, often, when users are in a state of confusion and not sure how to proceed using the software, they tend to gaze around the screen trying to find the best way to complete a task. This behavior is referred to as excessive effort. In this paper, the underlying theory of effort-based usability evaluation along with pattern recognition techniques are used to produce an innovative framework for the objective of identifying usability deficiencies in software. Pattern recognition techniques and methods are applied to data gathered throughout user interaction with software in an attempt to identify excessive effort segments via automatic classification of segments of video files containing eye-tracking results. The video files are automatically divided into segments using event-based segmentation, where a segment is the time between two consecutive keyboard/mouse clicks. Subsequently, data reduction programs are run on the segments for generating feature vectors. Several different classification procedures are applied to the features in order to automatically classify each segment into excessive and non-excessive effort segments. This allows developers to focus on the excessive effort segments and further analyze usability deficiencies in these segments. To verify the results of the pattern recognition procedures, the video is manually classified into excessive and non-excessive segments and the results of automatic and manual classification are compared. The paper details the theory of effort-based pinpoint analysis and reports on experiments performed to evaluate the utility of this theory. Experiment results show more than 40% reduction in time for usability testing.**

Keywords- Software Development; Software Usability; Human Computer Interaction; Pinpoint Analysis; Pattern Recognition; Clustering

I. INTRODUCTION

One of the primary goals of software is to simplify various tasks and enable users to accomplish tasks with ease and efficiency. Numerous fields have recently witnessed an increase in software development and deployment. Nevertheless, feedback from software applications end-users consistently shows that software is at times non-productive, confusing, counter-intuitive, and unsatisfactory [1]-[5]. Clearly, if the users experience problems or difficulty, it is highly unlikely that they will use that software again. Hence, it is very important for software engineers to place significant emphasis on usability evaluation and testing in order to eliminate user complaints and provide the user with a good experience [1]-[5].

Software engineers use a wide variety of tools, such as prototyping, inspection, usability testing, and iterative processes to ensure that the software they produce is usable [1]-[5]. Still, these tools may not address the usability problem efficiently, resulting in a low ranking on usability for several systems [1][5]. The classical methods used in identifying usability techniques have not proven to be very proficient in accurately locating the specific segment of code that could be leading to the usability problems. Without proper data to understand which part of code is faulty, developers would have a hard time identifying and fixing code that leads to usability issues.

The usability testing process involves observing users engaged with a software application and obtaining a set of characteristics of the user experience. This methodology requires an expert to construct, conduct, and assess the tests; as well as devoted laboratory facilities and several users that participate in the tests. Despite all of these efforts, generally usability testing indicates that a problem exists but does not identify the root cause for the problem [1][5]. This makes usability-testing time consuming, expensive, and frustrating for both developers and managers. Hence, it is often ignored.

Most of the tools used to evaluate the usability of a software application use 'time to complete a task', referred to as *time on task* (*ToT*), as a measure for evaluating usability [1]-[12]. This approach of giving high weight to *ToT* may not produce accurate results when factors like system performance, network delays, and interface design, which are difficult to avoid, play a role. An alternate approach is to measure usability in terms of user-effort, which eliminates some of the system issues mentioned earlier, allowing software engineers to focus on the interface design [1][5].

The *Effort-Based Usability Model* of [1][5][6][9]-[12] can be used for setting usability requirements, evaluating

user interface (UI), development of usable software, and pinpointing software usability defects. It is developed using the principle that usability is an inverse function of effort. The model is used for comparison of different implementations of the same application. The results of several experiments conducted on the effort-based model show a strong relationship between effort and usability [1][5][6][9]-[12].

The underlying theory of the *Effort-based Model* is used to produce a framework for identifying usability deficiencies in the software. Accurately locating software usability issues and correlating these issues with UI software code is referred to as Pinpoint Analysis [1][9]-[12]. For example, users who are in a state of confusion, and users that are not sure how to use the software, tend to look around the screen to determine the best way to accomplish a task. This behavior, which can be observed by eye tracking [13][14], is referred to as an excessive search or as excessive effort [1][5][9]-[12]. Identifying and pinpointing excessive effort behavior helps UI designers to rectify numerous usability related issues.

The hypothesis of this research is that it is feasible to devise a framework that can automatically identify *excessive effort segments* by applying pattern recognition techniques, such as K-means clustering algorithm, thresholding, principal component analysis (PCA), and feature selection [15]-[17]. Usability experts can further inspect the *excessive effort segments*. Hence, the automatic part can save experts' time and increase experts' accuracy.

To validate the hypothesis, this research, attempts to evaluate the utility of pinpointing UI deficiencies using pattern recognition techniques for identifying excessive effort in temporal segments of user software interaction. The process of segmentation of user's software interaction session and linking segments is done automatically using the time slice between two consecutive mouse/keyboard clicks. Automatic identification of segments with *excessive effort* behavior reduces the time required for UI designers to analyze and rearrange the interface at the pinpointed time snapshot. The last phase of the pinpoint process involves an expert evaluating *excessive effort segments*. Nevertheless, the process of identifying these segments is automatic and non-supervised.

The main contribution of this work is the development of a new methodology for assessing the usability of software. This methodology helps optimizing the time spent on usability testing while also more accurately identifying specific segments of code that could be leading to the usability issues.

Several experiments were conducted to validate the hypothesis and evaluate the new framework for pinpointing software usability issues. Experiment results show more than 40% reduction in time for usability testing.

The rest of this paper, which is an expanded version of [1], is organized as follows. Section II contains background information. Section III summarizes related work. Section IV presents the experimental setup. Section V details the experiments performed. Section VI presents experiment results and Section VII contains results evaluation. Section VIII concludes the paper with a summary of findings and proposals for further research.

II. BACKGROUND

A. Software Usability

According to the International Organization for Standardization/International Electro-technical Commission (ISO/IEC) 9126 standard, software usability is: "The capability of a software product to be understood, learned, used, and be attractive to the user when used under specified conditions" [8][9]. The standard lists several characteristics that play an important role in defining software usability: understandability, learnability, operability, and attractiveness [8][9]. The effort-based theory focuses on the first three characteristics.

Understandability helps determine how easy it is to comprehend and use the software. It is the ability of a user to understand the capabilities of the software and its suitability to accomplish specific goals. Learnability indicates the ease with which a user learns to use specific software. Operability is the capability of a user to use the software to accomplish a specific goal. Generally, the end-goal of a software application is to enable performing a task efficiently. As such, operability plays an important role in usability. Attractiveness relates to the requirement that the end-user's experience is pleasant and rewarding. The next section discusses several classical usability evaluation methodologies.

B. Classical Methods for Measuring Usability

The classical usability measurements methods are broadly classified into methods that make use of data gathered from users and methods that rely on usability experts. There are usability evaluation methods that apply to all stages of design and development, from product definition to final design modifications. Usability methods are further classified into *cognitive modeling methods*, *inspection methods*, *inquiry methods*, *prototyping methods*, and *testing methods*.

Cognitive models are based on psychological principles and experimental studies to determine times for cognitive processing and motor movements. They are used to improve user interfaces or predict problem areas during the design process. In general, cognitive modeling involves creating a computational model to estimate how long it takes for users to perform a given task [1]-[5]. It involves one or more evaluators inspecting a user interface by going through a set of tasks by which understandability and ease of learning are evaluated. The user interface is often presented in the form of a paper mock-up or a working prototype; but it might be a fully developed interface.

The inspection method involves cognition with emphasis on a hands-on approach. Under the inspection

method, experimenters observe users while they are using the software. The testing and evaluation of programs is done by an expert reviewer. This provides quantitative data, as tasks can be timed and recorded. In addition to quantitative data, qualitative user experience data are collected. Although some of the data collected is qualitative and potentially subjective, it provides valuable information [2]-[4].

Experts obtain information about users' likes, dislikes, needs, and understanding of the system by talking to them, observing them using the system, and through verbal or written questionnaires. Since this information is collected by inquiring and getting direct feedback from users, this model is called the inquiry method [1]-[5].

While the above methods focus on usability testing at an advanced stage in the development, the prototyping method tries to improve usability by refining and providing feedback as the software is being developed. Rapid prototyping is a method used in early stages of development to validate and refine the usability of a system. It is used to quickly and efficiently evaluate user-interface designs without the need for an expensive working model. This helps to remove the developer's resistance to design changes since it is conducted before any actual programming begins. Testing methods provide usability evaluation through testing of users for the most quantitative data. User interaction sessions are observed via two way mirrors or recorded on video that provides task completion time and allows for evaluation of user attitudes [1]-[5].

C. The Effort-based Usability Model

Several studies indicate that many system users associate the physical and mental effort required for accomplishing tasks with the usability of the software [1][5][6][9]-[12]. The *effort-based usability model* for software usability stems from the notion that the usability is an inverse function of effort. For example, an eye tracking device can be used to measure the effort expanded by the user in navigating through the user interface of software. According to the effort-based usability theory, the eye effort is inversely proportional to the operability of the software.

Physical and mental effort are obtained and inferred from logging user activity such as manual activities in the form of mouse movements and eye activities. For this model, *E* denotes the total effort required to complete a task with computer software and is defined as:

$$E = \begin{pmatrix} E_{mental} \\ E_{physical} \end{pmatrix}$$

$$E_{mental} = \begin{pmatrix} E_{eye\_mental} \\ E_{other\_mental} \end{pmatrix}$$

$$E_{physical} = \begin{pmatrix} E_{manual\_physical} \\ E_{eye\_physical} \\ E_{other\_physical} \end{pmatrix}$$

E_{eye_mental} denotes the amount of mental effort to complete the task measured by eye related metrics.

E_{other_mental} denotes the amount of mental effort measured by other metrics.

$E_{physical}$ denotes the amount of physical effort needed to complete the task.

$E_{manual\_physical}$ denotes the amount of manual effort required to complete the task. Manual effort includes, but is not limited to, the movement of fingers, hands, arms, etc.

$E_{eye\_physical}$ denotes the amount of physical effort *invested* in the process of interaction, measured by eye movement related metrics [13].

$E_{other\_physical}$ denotes the amount of physical effort measured by other metrics.

Consequently, the effort required to complete tasks is associated with software usability [1][5][9]-[12]. Physical effort includes manual effort and physical eye effort. In the case of interactive computer tasks, it is possible to calculate effort as a linear combination or a weighted sum of metrics such as the number of mouse clicks, number of keyboard clicks, *average eye path traversed* as well as other eye activity measures, and *mouse path traversed* [1][9]-[12].

Mental effort is essentially the amount of brain activity required to complete a task. To some extent, brain activity, related to a task, can be approximated by processing eye movement data recorded by an eye tracker [1][5][9]-[14]. Eye trackers acquire eye position data and enable classifying the data into several eye movement types useful for eye related effort assessment. The main types of eye movements are [13][14]:

1) *Fixation* – eye movement that keeps an eye gaze stable with regard to a stationary target providing visual pictures with high acuity. Fixations might be a result of "interest" or a result of confusion. In the context of task completion, fixations are generally correlated to confusion.

2) *Saccade* – rapid eye movement from one fixation point to another.

3) *Pursuit* – stabilizing the retina with regard to a moving object of interest. Usually, however, the Human Visual System (HVS) does not exhibit pursuits when dynamically moving targets are not a part of the interface [13][14].

In addition, some eye trackers supply information about ToT as well as user manual activity including mouse and keyboard clicks.

In this research, we concentrate on the correlation between physical effort and usability. The following metrics, which have been identified as the most important effort-based metrics, are used as a measure of the *physical* effort [1][5][6][9]-[14]:

1) *Average fixation duration*,
2) *Average saccade amplitude*,
3) *Number of fixations*,
4) *Number of saccades*, and
5) *Average eye path traversed*.

Additional commonly used metrics such as the number of keyboard clicks and the number of mouse clicks are used for identifying segments of interaction rather than classifying segments.

The effort-based software usability evaluation is divided into three phases: *Measurement, Analysis, and Assessment* [1]. In the *Measurement phase*, a group of users executes a set of tasks referred to as identical independent tasks, due to the fact that they share characteristics with an identical independent distribution (iid) used in probability theory. The tasks emerge from a single scenario; however, several parameters change from task to task in a pseudo random fashion. Hence, these tasks differ in key parameters, which prevent the users from memorizing a sequence of interaction activities. Throughout the interaction process, certain user actions such as eye movement, *ToT*, keyboard activities, and mouse activities are logged.

The *Analysis* phase involves accumulating data for several physical effort-based metrics such as the *number of saccades*, *average saccade amplitude*, *number of fixations*, *average fixation duration*, and *average eye path traversed*. Another metric is the *ToT*. The average task completion time and/or an effort-based metric are compared to a learning curve, which reflects users' mastery of software.

The final step is the *Assessment*. Using the above steps, the learnability of software systems is assessed and the point of users' mastery of the software is identified. In addition, as detailed in section D, the learnability curve is used to obtain operability and understandability of various software systems or different groups of users using the same system. Effort-based metrics provide interface designers with means to evaluate their designs [1][5].

D. The Learnability based Usability Model

Typically, as users become familiar with an application, the effort and/or the time to complete tasks, which emerge from the same scenario, become smaller or shorter [18]. Often, a graph of the averages of *Effort-On-Task* (*EoT*) or *Time-On-Task* (*ToT*), also known as effort average (E_{avg}), for the users fits well into an exponential decay curve that represents the average effort on task expended by the group of users. Figure 1 depicts a typical graph. The E_{exp} line is the effort that the interface designer expects an expert to expend in order to complete a specific task. The point where the user's effort reaches the acceptable level is the learning point L_p. The learning time (L_T) is calculated by adding the average task duration to the left of the learning point. Data to the right of the learning point relates to the amount of effort required by a trained user to complete tasks.

Understandability can be inferred from the graph by investigating the difference between the expert curve and the average user curve; while operability can be inferred from the distance between the expert curve (E_{exp}) and the *X*

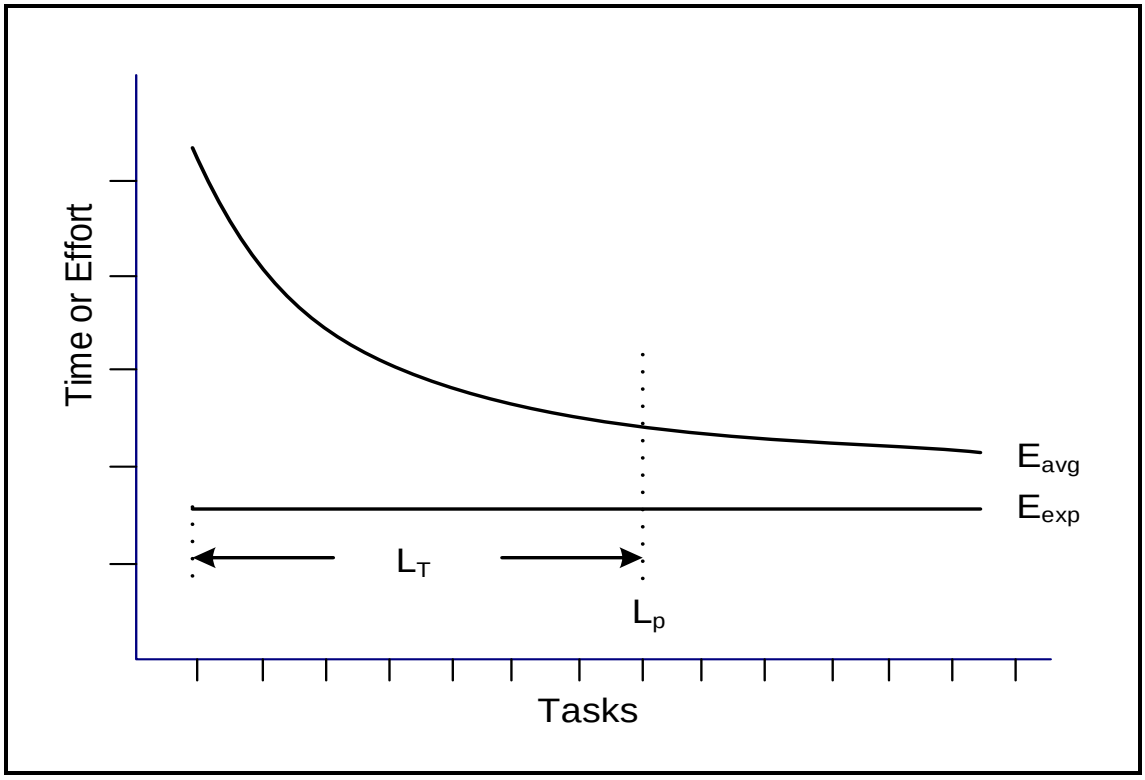

Figure 1. Learnability-based usability model

axis. Thus, the *effort-based usability model* enables the evaluation of the understandability learnability, and operability of a specific system. In addition, comparing the plots representing the results of tests with different systems or different user populations (e.g., students vs. novice employees) can be used to evaluate the relative usability of these systems [1]. This is referred to as "system A vs. system B" or "population A vs. population B" experiments [1]. Moreover, this model is further used to identify outlier tasks, which are studied to find usability shortfalls [1]. Outlier tasks are good candidates for a specific type of pinpoint analysis referred to as inter-pinpoint analysis.

E. Pinpoint Analysis

Software usability testing is one of the most expensive, tedious, and least rewarding tests to implement [1]-[5]. This perception is likely to change if the usability testing is made less expensive and more rewarding. This requires accurate means through which an engineer can identify and pinpoint issues in the software or the interface. This process is called pinpoint analysis. Pinpoint analysis is one of two types; inter-pinpoint analysis deals with identifying issues with tasks performed by the users in a specific system, whereas intra-pinpoint analysis refers to identifying issues within tasks in a specific system. For example, outlier tasks might be identified through inter-pinpoint analysis and used for intra-pinpoint analysis. This analysis can help graphical user interface (GUI) designers to make decisions about element placement on displays and determine the level of effort that is related to different widgets [1][5].

1) Inter-pinpoint Analysis

Inter-pinpoint analysis involves detecting tasks that present anomalies and identifying the reasons for these anomalies at a high level. The mouse is used as an example to illustrate inter-pinpoint analysis. In a particular task, the right mouse button helps users complete a task effectively; however, some of the users are unaware of it. It is possible that anomalies like this can be identified in inter-pinpoint analysis [1][5].

Inter-pinpoint analysis helps in identifying alternative methods to perform a task effectively with less effort; however, it does not provide users with a hint of the alternative method. Other issues like the necessity of help facilities in software are identified by the high level analysis of tasks that present anomalies.

Figure 2 is a plot of the average ToT of five subjects for seven identical but independent tasks. The X axis shows a data point for each of the seven tasks, while the Y axis shows information related to the ToT. The curve fitted to the individual bars representing the average ToT is an exponential curve that actually corresponds to the learnability model. The high correlation value ($r^2 = 0.96$) shows that the exponential curve well fits the data. Again, placing the plot of more than one systems' test in the same graph can be used for a "system/population A vs. system/population B" comparison. In this case, task-3 that does not fit well in the curve shows an anomalistic behavior and calls for further analysis and study [1][5].

2) Intra-pinpoint Analysis

Intra-pinpoint analysis is a detailed method for analyzing tasks and identifying specific issues with the software. The analysis can be done manually by watching video recordings of users' interactions with software and/or watching videos obtained from an eye-tracking device. The review helps in identifying interaction issues and areas where the user has difficulty while performing tasks. For example, the analysis might reveal that most of the users go into a state of confusion in a specific part of a task, and are searching the screen to identify the best way to proceed with

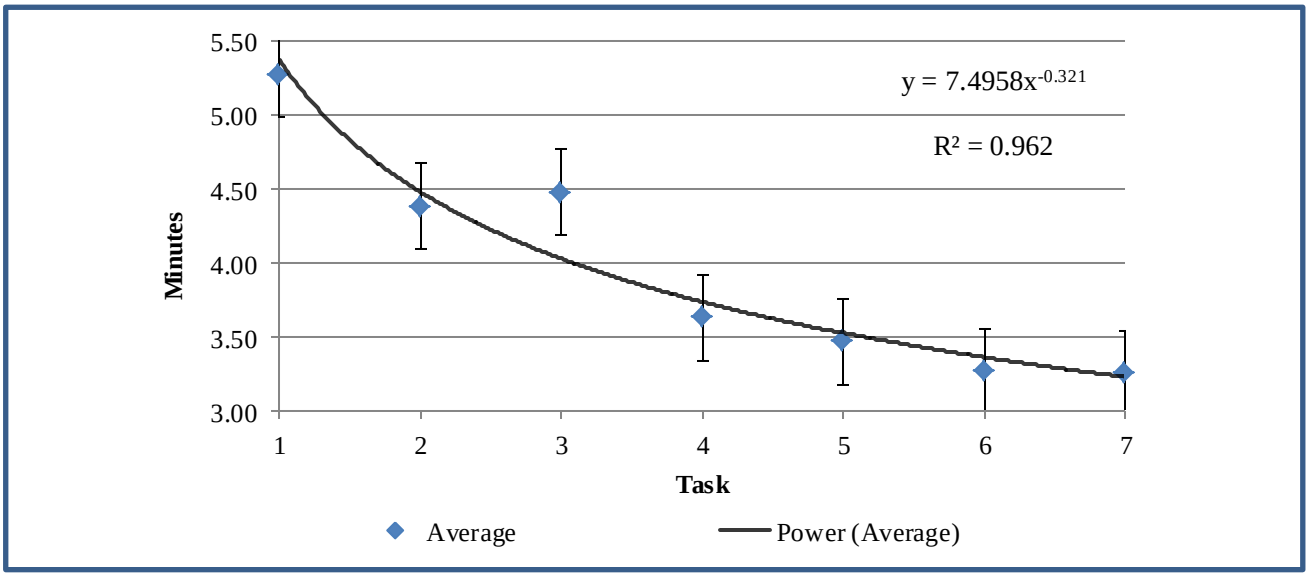

Figure 2. – Time-on-Task (TOT) for the use case of interest.

the task. This might prompt the designers to rearrange the interface where a snapshot identifies excessive effort. Clearly, the manual recording inspection is tedious and potentially expensive. An alternative is to use automatic methods utilizing pattern recognition techniques. This method eliminates the need for a person to watch the entire video in order to identify interaction issues, thereby cutting down the cost and time. It enables automatic identification of areas where the user has difficulty and marking these areas for further evaluation.

F. Pattern Recognition

One of the applications of pattern recognition is the assignment of *label*s to a given input value, or *instance*, according to a specific algorithm. An example of pattern recognition procedure is classification, which attempts to assign each input value to one of a given set of classes.

Pattern recognition is generally categorized according to the type of learning procedure used to generate the output value. *Supervised learning* assumes that training data (the training set), consisting of a set of instances that have been properly manually labeled by an expert with the correct output, has been provided. Next, a learning procedure generates a model that attempts to meet two, sometimes conflicting, objectives: Perform as well as possible on the training data, and generalize as well as possible to new data. On the other hand, *unsupervised learning* assumes the availability of training data that has not been hand-labeled and attempts to find inherent patterns that are used to determine the correct classification value for new data instances [15]-[17].

Algorithms for pattern recognition depend on several parameters, such as the type of output labels, and on the training/learning methods that are supervised or unsupervised. Additionally, the algorithms differ in the way that inference is performed. For example, inference might be based on probability, non-parametric clustering, fuzzy logic, etc. [15]-[17]. The following are various relevant pattern recognition techniques.

1) Segmentation

Pattern recognition procedures require the definition of patterns (i.e., segmentation). In this research, segments of user activities records serve as the basic patterns. A segment is defined as the time between two consecutive keyboard/mouse clicks.

2) Feature Extraction and Feature Selection

Generally, the objects that are subject to classification, i.e., the patterns (segments in the case of this research), are represented through a set of measurements (say n measurements) or characteristics referred to as features. Hence, the objects are considered as vectors in an n-dimensional space referred to as the feature space. Feature selection is a technique for selecting a subset of relevant features for building robust learning and inference models [15][16]. Feature selection algorithms attempt to reduce the dimensionality of the feature space and reduce the complexity of the recognition process by pruning out redundant, correlated, and irrelevant features. There are several feature selection algorithms, some of which are discussed below [16].

Exhaustive search is a brute-force feature selection method where all possible subsets of the features are exhaustively evaluated and the best subset is selected. The number of combinations of r objects from a set of n features is $(\frac{n!}{r!(n-1)!})$. This might result in a very large set of combinations of features to examine. Hence, generally the exhaustive search's computational cost is prohibitively high. Thus, this method is impractical if the number of features in the subset is large or the processing and evaluation time for each subset is long [11][16]. Because of the problems associated with exhaustive search, researchers resort to adopting heuristic feature selection algorithms. In this paper, there are five features of interest. This is a relatively small number of features. Nevertheless, each evaluation session requires significant computation time. Hence, due to the complexity of the evaluation process, exhaustive search is not a viable option. For these reasons, a heuristic approach is adapted.

Heuristic search refers to selecting a feature subset by making an educated guess and finding out if the selection yields good results. Otherwise, the heuristic procedure examines other subsets [16].

3) Principal Component Analysis

PCA is an unsupervised regression procedure that analyzes data samples, such as the set of training patterns, in order to identify a coordinate transformation that de-correlates the data and "orders" the information (or variance) associated with the data in the axes of the new space in a monotonically non-increasing fashion. In general, as a result of the transformation, most of the information associated with the data is concentrated in the first few components of the new space. This enables ignoring components (axes) that do not carry significant information, thereby reducing the dimensionality of the space used for pattern representation and recognition. Each principal component is a linear combination of the original variables. The principal components as a whole form an orthogonal basis for the data space [16].

The distinction between PCA and feature selection is that following the PCA the resulting features are different from the original features; they do not correspond directly to the set of measurements, and are not easily interpretable, while the features left after feature selection are simply a subset of the original features.

Following the feature selection and/or PCA, classification is applied via different methods including thresholding, discriminant analysis, decision functions, and clustering [15]-[17].

In this research, heuristic based greedy feature selection techniques as well as PCA are used to reduce the dimensionality of the data set consisting of a number of interrelated variables, such as the *number of saccades* and the *average saccade amplitude*, *number of fixations* and *average fixation duration* while retaining as much as possible of the variation present in the data set. In the case of PCA, this is achieved by transforming the data set into principal components, The principal components are then subjected to thresholding and/or clustering algorithms to find segments of excessive effort.

4) The Threshold Method

The threshold method can be used to classify input data based on a threshold value. In this research, the threshold value for each feature is the average of the values of the feature over the entire set of segments. All values greater than the threshold are placed into the "excessive effort" group while input values below the threshold are placed into the "*non-excessive effort*" group. One problem with the threshold method is that it is limited to one dimensional data. Hence, it is only applied to individual features, or a combination of features, such as linear combination or a specific component of the principle components. Clustering techniques, however, are used to efficiently classify multidimensional data.

5) Clustering

Clustering is a multi-disciplinary, widely-used, unsupervised method for classifying data. It involves the assignment of a set of patterns into subsets (called clusters) so that patterns in the same cluster are similar in some sense. To define a cluster, it is necessary to first define a measure of similarity or distance, which establishes a rule for assigning patterns to the domain of a particular cluster center. Generally, and in this paper, Euclidian distance is used as the distance measure. In Cartesian coordinates, if $p = p_1, p_2, \ldots, p_n$ and $q = q_1, q_2, \ldots, q_n$ are two points in an n dimensional space, the Euclidean distance between p and q is:

$$D(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

The Euclidean distance is used as the measure of similarity; the smaller the distance, the greater the similarity. There are several clustering algorithms, such as the hierarchical, partitional, density based, and subspace clustering algorithms [15]-[17]. In this research, however, partitional algorithms are of interest. Partitional clustering involves partitioning of n observations (patterns) into k clusters where each observation belongs to the nearest cluster. The K-means algorithm, used in this research, is a partitional algorithm that attempts to minimize the mean square distance between patterns and cluster centers, where the cluster center is the centroid of the cluster patterns. The algorithm consists of the following steps [15]:

Step 1 (seeding): Choose K initial cluster centers $z_1(1), z_2(1), \ldots, z_k(1)$. The seeds can be chosen in many different ways[13][15]. In this research, K random centers serve as seeds. The set of cluster centers at the L^{th} iteration is denoted by $z_i(L)$, where, $i = 1,2, \ldots, K$.

Step 2: At the M^{th} iterative step, distribute the patterns $\{x\}$ among the K cluster domains, using the following decision rule $x \in S_j(M)$ if $\|x - z_j(M)\| \leq \|x - z_i(M)\|$ for all $i = 1,2, \ldots, K$, $i \neq j$, where $S_j(M)$ denotes the set of patterns whose cluster center is $z_j(M)$.

Step 3: Using the results of step 2, compute the new cluster centers, such that the sum of the squared distances from all points in $S_j(M)$ to the new cluster center is minimized. The new cluster centers are given by

$$z_j(M+1) = \frac{1}{N_j} \sum_{x \in S_j(M)} x$$

For $j = 1,2, \ldots, K$, where N_j is the number of samples in $S_j(M)$.

Step 4: Repeat step 2 and step 3 until there is no change in the cluster centers, i.e., if $z_j(M+1) = z_j(M)$ for $j = 1,2, \ldots, K$, then the algorithm has converged and the procedure is terminated.

The advantage of the clustering technique is its ability to classify *excessive effort segments* by considering a number of features such as *saccade count*, *average saccade amplitude*, *fixation duration*, *and average eye path traversed*. In addition, the K-means clustering can be used to identify thresholds. MATLAB, a high-level programming language and interactive environment for numerical computation, visualization, and programming [19], has a built in function for K-means that operates exactly as described in this section. This function has been used in our project.

III. LITERATURE REVIEW

Usability is a highly researched topic with much literature available [1-6][21]-[23]. Nevertheless, extensive review did not reveal any research papers related to pinpointing usability issues (except for our prior work described in [1][9]-[12]. There are some papers on effort-based usability evaluation that are discussed below.

Tamir et al. concluded that effort and usability are related but they did not address pinpointing issues [5]. Mueller et al. use effort metrics to evaluate software usability [6]. Their method allows comparison of two or more implementations of the same application, but does not identify where exactly the problem lies. Hvannberg et al. described the design and test of a defect classification scheme that extracts information from usability problem [20], but is limited since it does not define the causes underlying usability problems. Nakamichi et al. investigate the relations between quantitative data, viewing behavior of

users, and web usability evaluation by subjects [21]. They conclude that the moving speed of the gazing points is effective in detecting low usability. Makoto et al. use a Web-Tracer to evaluate web usability [22]. Web-Tracer is an integrated environment for web usability testing that collects the operation log of users on the Web pages. However, the reasons for low usability are not identified using this approach. Our paper thoroughly addresses and resolves all of the issues listed above.

IV. EXPERIMENTAL SETUP

A. Manual Input Devices

The subject performs the tasks on a computer using a standard keyboard and a mouse as input devices. An event driven logging program is used to obtain details of mouse and keystroke activities from the operating system event queue. The program saves each event along with a time stamp into a file. The logged events are: mickeys (mouse pixels), keystrokes, mouse button clicks, mouse wheel rolling, and mouse wheel clicks.

The eye tracker used for the experiments is Tobii X120 Eye Tracker with Tobii Studio version 2.2.5 [23] as well as "in-house" developed software for estimating user's gaze. The Tobii device is a standalone eye tracking unit designed for eye tracking studies. It provides raw eye gaze positional data and is able to log mouse and keyboard events. The data collected by the eye tracker is logged into a file, which is referred to as a *log file*. The eye tracker also records a video version of the user interaction session and is referred to as a *video file*, which is very helpful in verifying experiment results. In addition, the Tobii X120 eye tracker can log mouse/keyboard clicks. The combination of the *log file* and *video file* are referred to in this paper as the *data file*.

B. Software Environment for Analysis

A software program developed in MATLAB is used to perform data analysis of the experiments reported in this paper [19]. In addition, the program is responsible for features collection and extraction.

C. Test Procedure

Experiments conducted to evaluate the capability of pattern recognition techniques to identify software usability issues are done using the steps depicted in Figure 3. As the figure shows, the main steps are: data gathering, segmentation, data reduction, feature extraction and selection. These actions are followed by several different classification techniques. The sequence of actions depicted in the figure is further described in the next three subsections: data gathering, data reduction, and identification of *excessive effort segments*.

1) Data Gathering

A group of five users executes a set of seven identical independent tasks, which emerge from a single scenario. Throughout the interaction process, certain user activities such as eye movement, ToT, keyboard, and mouse activities are logged using the eye-tracking device. According to the *learnability-based usability model*, the point at which the user's effort reaches the acceptable level is called the learning point. Based on this model, and inspection of the learning curve of the subjects, it is assumed that the users' effort reaches the acceptable level by the time they perform task-5. Hence, in this paper, task-5 of each subject is used for conducting the pinpoint analysis experiments.

2) Data Reduction

Phase-2 includes activities such as segmentation, data reduction, and feature extraction. The data logged throughout the user interaction session is used for automatic event based segmentation where the events are consecutive keyboard/mouse clicks. Metrics such as:

(a) *segment duration* (for event based segmentation),
(b) the *average fixation duration*,
(c) the *average saccade amplitude*,
(d) the *number of fixations*,
(e) the *number of saccades*, and
(h) the *average eye path traversed*

are inferred for each segment. These metrics are used to generate a feature set, which is obtained by applying data reduction programs to the data file. The features data is calculated for all features within each segment and this data is used to identify *excessive effort segments*.

2) Identification of Excessive Effort Segments

Pattern recognition techniques are applied to the feature set obtained from the data reduction process to identify segments that exhibit excessive effort. The techniques used and applied on the feature set are thresholding, K-means clustering, and PCA.

Thresholding - a threshold value is calculated for each feature in the feature set. For a given feature, all the segments that have a feature value that is less than the threshold value are classified as *non-excessive effort* segments and segments with a feature values above the threshold are considered as *excessive effort segments*. In the current research, the threshold is the average value of the feature in the segment.

K-means clustering - the segments are grouped into clusters. Based on the value of cluster centers, the cluster is classified as excessive effort cluster or non-excessive effort cluster.

PCA - the first, the second, and the third principal components of the feature data are obtained. The threshold classification, where the threshold is the average value of the first principle component computed over the set of features extracted for the current segment, is applied on the *first principal component* and K-means clustering is applied on the first, second, and third components to classify the segments into *excessive effort* or *non-excessive effort segments*.

By the end of phase-3, the software program identifies the *excessive effort segments*. To verify the results, the video

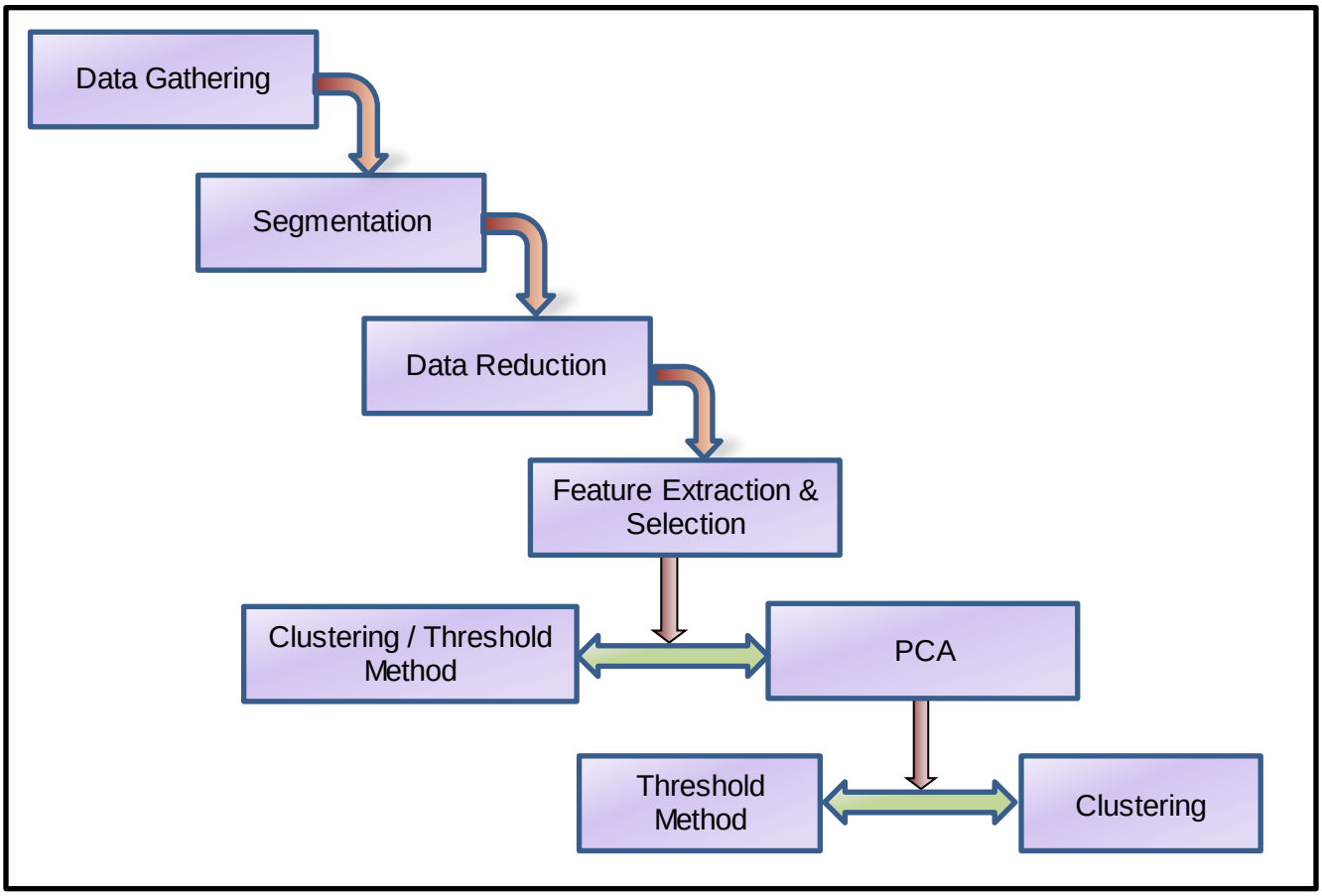

Figure 3. Experiment procedure

file is carefully watched segment by segment and classified into *excessive* or *non-excessive effort segments* manually. The manual analysis results are used as *ground truth* and serve as the input for error analysis that further supports the reliability of our results. The manual classification process of the video file is described in the following section.

D. Manual Classification

The manual classification process involves automatic event based segmentation on the entire video file. Each segment is carefully watched and classified into the following categories:

Idle behavior segments: Idle behavior is due to system response. Waiting for a progress bar to complete or waiting for a page to load are examples of idle behavior. Segments with such behaviors are classified as idle behavior segments.

Excessive effort segments: Segments without any useful user actions are classified as *excessive effort segments*. A subject looking at different components on an interface instead of the actual target component, which help in accomplishing the task, is an example of excessive effort behavior. Such behavior can be eliminated without sacrificing task completion quality.

Non-Excessive effort segments: Segments with useful action that result in task completion are classified as non-excessive segments.

Off screen behavior segments: Intervals of time where the subject's view is not within the screen for more than one second, with no meaningful user action, are classified as off screen behavior segments.

Attention segments: Segments with frequent on screen behavior, e.g., segments with very frequent mouse/keyboard clicks are classified as attention segments.

Once the video file is classified into one of the above five segment categories, the manual classification results are compared with the automatic classification results. Nevertheless, idle behavior, off screen behavior, and attention segments can be accurately identified by the software tool and are discarded from further analysis in this work. In this sense, our results are conservative as we so not measure the additional time saving obtained via the identification these types of segments.

E. Result Verification

The number of *Excessive* (*E*) *vs. Excessive*, *Excessive vs. Non-Excessive (NE)*, *NE vs. E* and *NE vs. NE* segments,

as well as related error rates, are calculated for each result file and graphs are plotted to visualize the results and enable comparing the performance of different methods and features. Classifying *NE* segments as *E* segments is regarded as false positive or type-I error.

It is assumed that all the segments classified as *Excessive Effort Segments* are due for an additional process of manual evaluation. Hence, in the case of type-I error, the software program is highlighting extra segments for further review but is not missing segments that need attention.

On a similar note, segments that show excessive effort per manual classification but are identified as *non-excessive effort segments* by the software program are regarded as false negative or type-II error segments. These segments require extra attention as the software program has misidentified segments that require the stage of manual inspection. The total time of segments classified as excessive by the software program is referred as the inspection time. It is the sum of the time interval of each of the *excessive effort segments* automatically identified by the program. In this research, type-II errors and inspection time are considered as the most important factors for analyzing experiment results.

V. EXPERIMENTS

The automatic part of the process is used to analyze the five data files by applying the different pattern recognition techniques discussed. Each of the data files is a *log file* (log of effort metrics expanded) and an eye tracking video file that contains the entire data collected by the eye tracker throughout each experiment. The following is a list of the classification experiments performed: 1) Applying the threshold method, 2) Applying heuristic feature selection and K-means clustering, 3) Using PCA, and 4) Applying K-means clustering on principal components.

Each experiment procedure is discussed in detail in the following sections.

A. Experiment 1: Applying the threshold method

In this experiment, automatic event based segmentation is applied to the eye tracking video and data file generated by the eye tracker. Next, a feature set is generated for the data file. All the segments are classified into excessive or *non-excessive effort segments* by the software program, which applies the threshold method on the following features: 1) *number of fixations*, 2) *average fixation duration*, 3) *number of saccades*, 4) *average saccade amplitude*, and 5) *average eye path traversed*.

Figure 4 presents the sequence of steps for identifying *excessive effort segments* using the threshold method.

The steps described in Figure 4 are used for identifying the *excessive* and *non-excessive effort segments*. Next, the video file segments are manually classified into *excessive* or *non-excessive effort segments* based on the specifications described above. The *excessive effort segments* identified through the software program and manual process are verified using the five data files and their corresponding video files. This step of manual classification and result verification is done in each of the experiments described in this section.

B. Experiment-2: Applying heuristic feature selection and K-means clustering

The evaluation process of a subset requires a long execution time. Hence, evaluating all the possible subsets of the feature set is prohibitively time consuming, we have adopted a heuristic greedy-based feature selection method. The following subsets have been selected: 1) *Number of fixations*, 2) *Number of saccades*, 3) *Average eye path traversed*, 4) *Number of fixations, number of saccades*, and *eye path traversed*, and 5) *Number of fixations, number of saccades, eye path traversed, average fixation duration* and *average saccade amplitude*.

Figure 5 illustrates the sequence of steps followed in identifying *excessive effort segments* using exhaustive feature selection and K-means clustering.

C. Experiment-3: Using PCA.

In this experiment, the feature set is transformed into principal components by a MATLAB function. Only the *first principal component* is considered, as it carries the most significant information related to the feature set. The *first principal component* is subjected to the threshold method for identifying segments exhibiting *excessive effort* and *non-excessive effort*. Figure 6 depicts the sequence of steps applied for identifying excessive effort, the method

Figure 4. Sequence of steps for identifying excessive effort segments using the threshold method

for identifying segments exhibiting *excessive effort*, and *non-excessive effort*.

D. Experiment-4: Applying K-means clustering on principal components

In this experiment, K-means clustering is applied to different combinations of principal components for identifying segments exhibiting *excessive effort* and *non-excessive effort*. The following constitute the feature set for this experiment: 1) 1st principal component, 2) 1st and 2nd principal components, and 3) 1st, 2nd, and 3rd principal components.

Figure 7 includes a diagram of the sequence of steps followed for identifying *excessive effort segments* using the K-means clustering on principal components.

VI. RESULTS.

In this section, the results obtained from the experiments are discussed. The results of each data file in the experiments are given in [11]. A sample of these results is presented here. For clarity, the notation used for the feature values in the graphs is presented below:

1) # Fix – denotes the *number of fixations*,
2) Avg. Fix Dur. – denotes the *average fixation duration*,
3) # Sacc – denotes the *number of saccades*,
4) Sacc Amp. – denotes the *average saccade amplitude*,
5) Eye Path - denotes the *average eye path traversed*,
6) FPC - denotes the first principal component:

A. Identifying excessive effort segments using the threshold method

In this section, we show the results obtained with data file-1. Results with other files are available in [11]. Section VII shows and analyzes the average results for all the files. The video file corresponding to data file-1 is 6.09 minutes in length. Figure 8 shows the results of an experiment using the threshold method on data file-1.

When the graph in Figure 8 is extrapolated and as seen from the *E vs. NE* bars, the feature value, *number of*

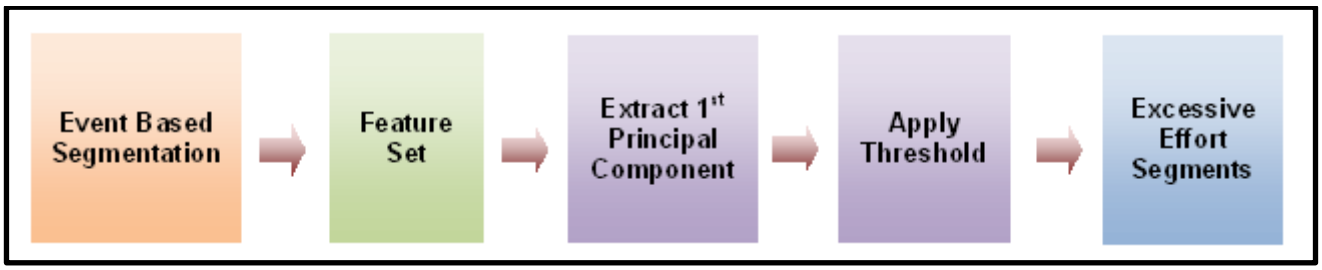

Figure 5. Sequence of steps for identifying excessive effort segments using the K-means clustering.

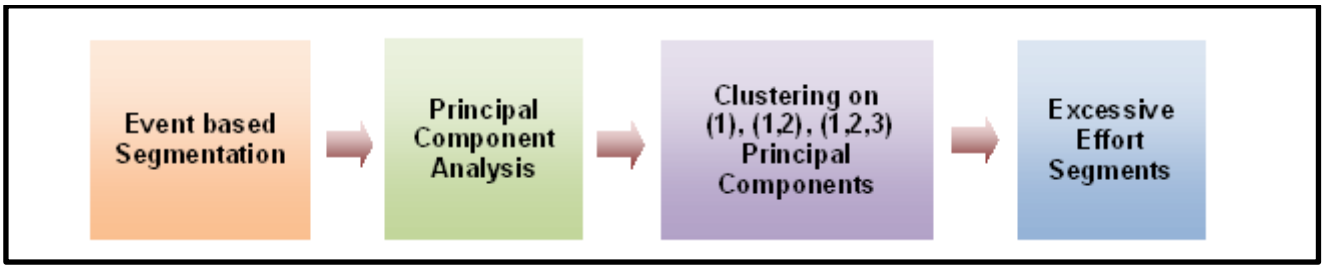

Figure 6. Sequence of steps for identifying excessive effort segments using the threshold method on the first principal component.

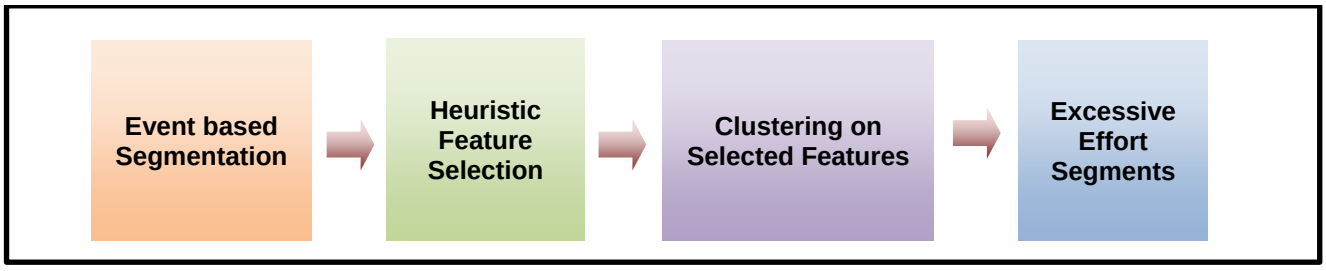

Figure 7. Sequence of steps for identifying excessive effort through K-means clustering on the 1st principal component.

fixations, demonstrates a small percentage of *E vs. NE* segments. This shows that the *number of fixations* has the least number of type-II errors. *Number of saccades* and *average eye path traversed* follows the *number of fixations* in terms of type-II errors.

Figure 9 shows the total time of segments classified as excessive by the software program and the manual process after the threshold method is applied on each of the following features: 1) *number of fixations*, 2) *average fixation duration*, 3) *number of saccades*, 4) *average saccade amplitude*, and 5) *average eye path traversed*.

The light black bars in Figure 9 represent the total video time recorded by the eye tracker. Manual classification of the video file, depicted by the dark black bars, shows 1.71 minutes of excessive effort. The *average fixation duration* and *average saccade amplitude* show a relatively low value for time of segments classified as excessive by the software program when compared with the total video time. This is depicted by the bright bars present in the figure. From Figure 9, it is observed that the percentage of type-II errors is 15.05% for *average fixation duration* and 12.9% for *average saccade amplitude*. However, the feature value with a reasonable type-II errors and lower percentage of time of segments classified as excessive is *average saccade amplitude*.

It should be noted that we are considering a 15% error of type-II as acceptable. This is explained in section VII.

B. *Identifying excessive effort segments using heuristic feature selection and K-means clustering*

In this section, we show the results obtained with data file-2. Results with other files are available in [11]. The

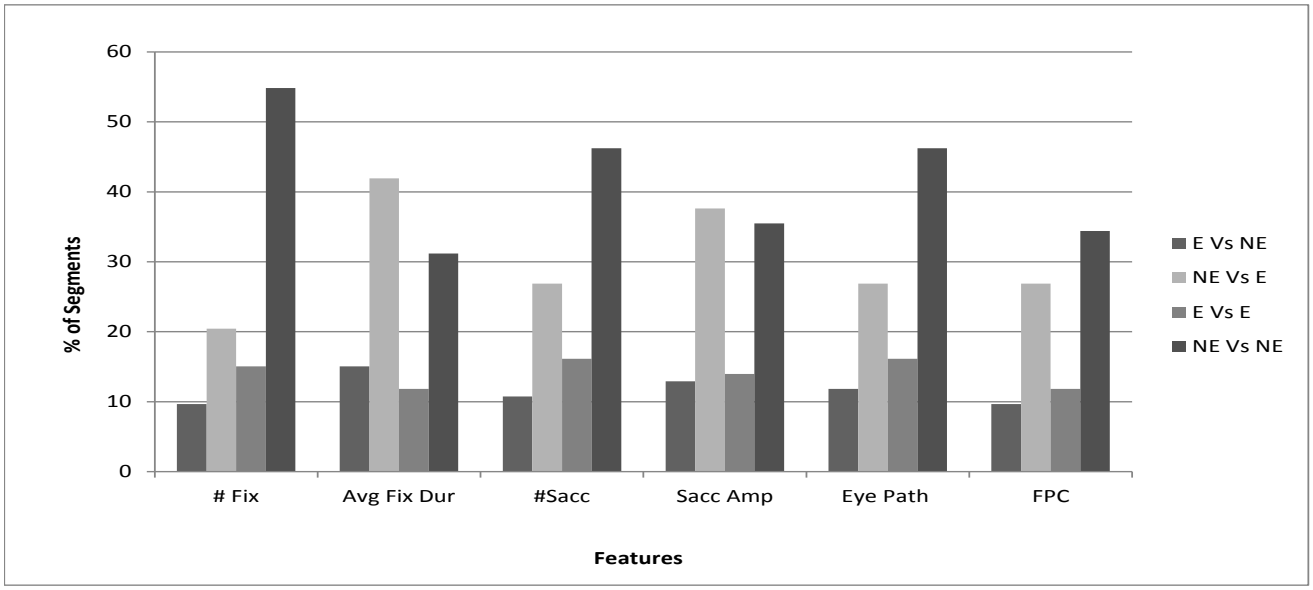

Figure 8. Percent of segments of each type (file-1, experiment-1).

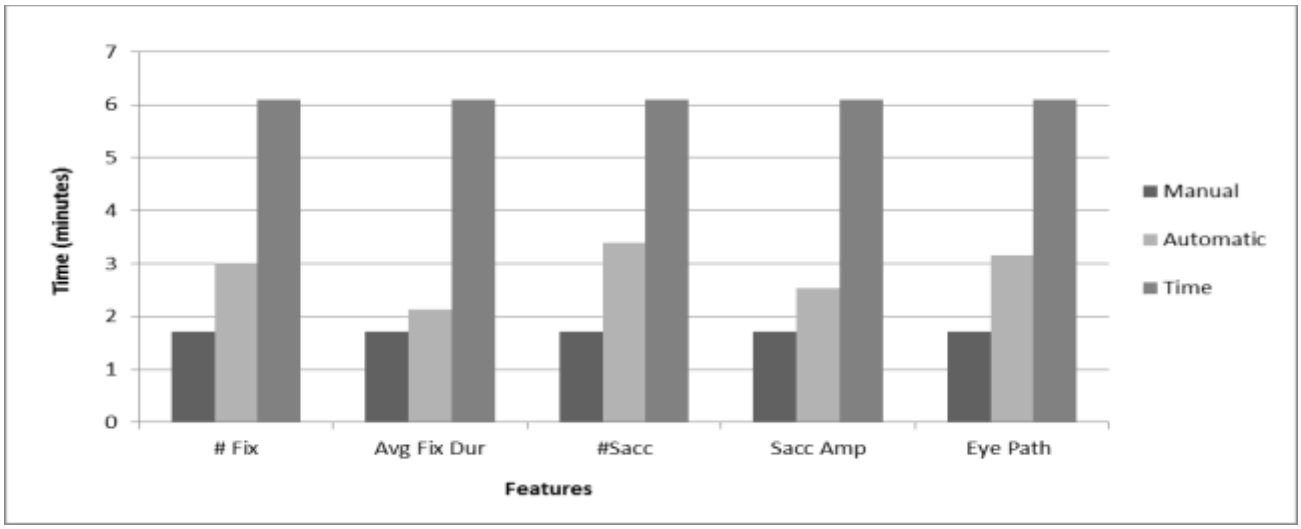

Figure 9. Total time of excessive effort segments (file-1, experiment-1).

video file corresponding to data file-2 is 3.27 minutes in length. Figure 10 shows the results of an experiment using the K-means clustering on data file-2.

When the graph in Figure 10 is extrapolated and as seen from the *E vs. NE* bars, feature subset-1 (defined above) demonstrates a small percentage of *E vs. NE* segments. This shows that the subset-1 has the least number of type-II errors. Subset-2 follows subset-1 in terms of type-II errors.

Figure 11 shows the total time of segments classified as excessive by the software program and the manual process for the above defined five feature subsets.

The light dark bars in Figure 11 represent the total time of video recorded by the eye tracker. Manual classification of the video file, depicted by the brightest bars, shows 0.36 minutes of excessive effort. Subset-3 shows a relatively low value for time of segments classified as excessive by the software program when compared with the total video time. This is depicted by the dark bars in the graph. From Figure 11 it is observed that the percentage of type-II errors is 8.5% for subset-3. Therefore, the feature value with an acceptable error of type-II and lower percentage of time of segments classified as excessive is subset-3.

C. Identifying excessive effort segments using PCA

The results of all the data files are consolidated into a single graph. Figure 12 shows the percentage of segments of each type when applying the threshold method on the *first principal component* for all five data files.

From the graph in Figure 12, it is clear that using the threshold method on the *first principal component*s produces a small percentage of *E vs. NE* segments. This means a lower type-II of errors as seen from the respective bars in the graph.

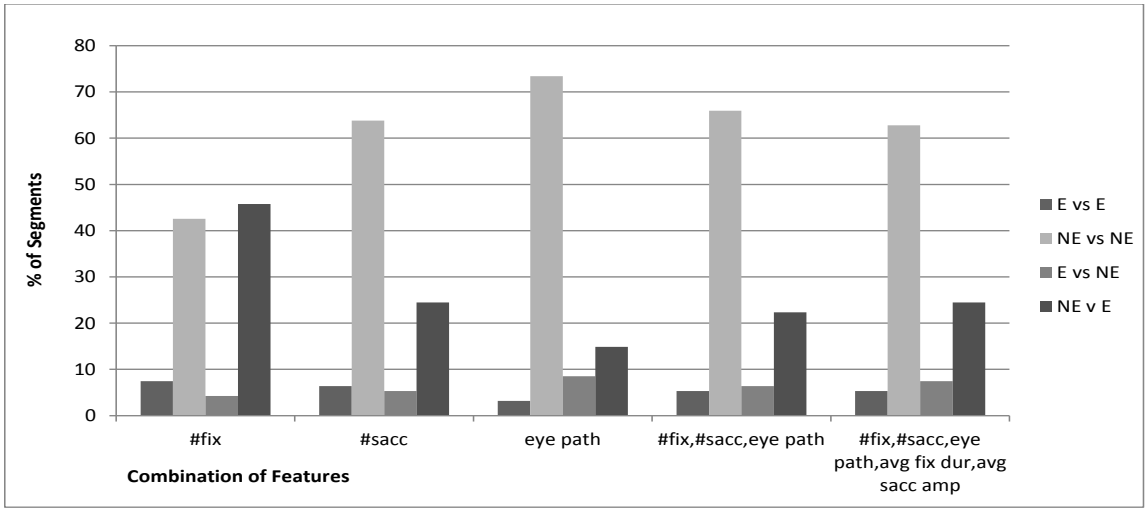

Figure 10. Percent of segments of each type (file 2, experiment-2).

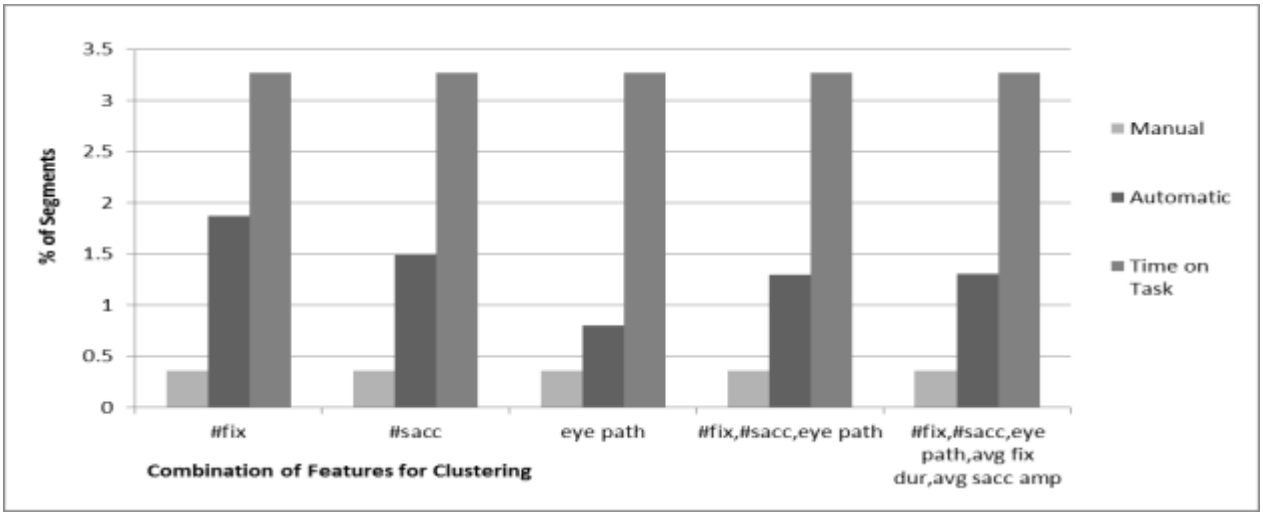

Figure 11. Total time of excessive effort segments (file 2, experiment-2).

Figure 13 shows the total time of segments classified as excessive by the software program and the manual process for the *first principal component*. The results of all five data files are plotted in a single graph.

The "light dark" bars in Figure 13 represent the total video time recorded by the eye tracker for each of the five data files. Manual classification of the video files is depicted by the dark bars. The bright bars represent the total time of video classified as excessive by the software program. The percentage of time of segments classified as excessive is relatively high when applying thresholding on the first principal component.

D. *Identifying excessive effort segments using K-means clustering on principal components*

In this section, we show the results obtained with data file 3. Results with other files are available in [11]. The video file corresponding to data file 3 is 3.8 minutes in length. Figure 14 shows the percentage of *E vs. E, E vs. NE, NE vs. NE, and NE vs. E* segments for the features mentioned in the experiment description.

From the graph in Figure 14, it is clear that all three features have same percentage of *E vs. E*, *E vs. NE*, *NE vs. NE* and *NE vs. E* segments. This signifies that most of the information is concentrated in the *first principal component*

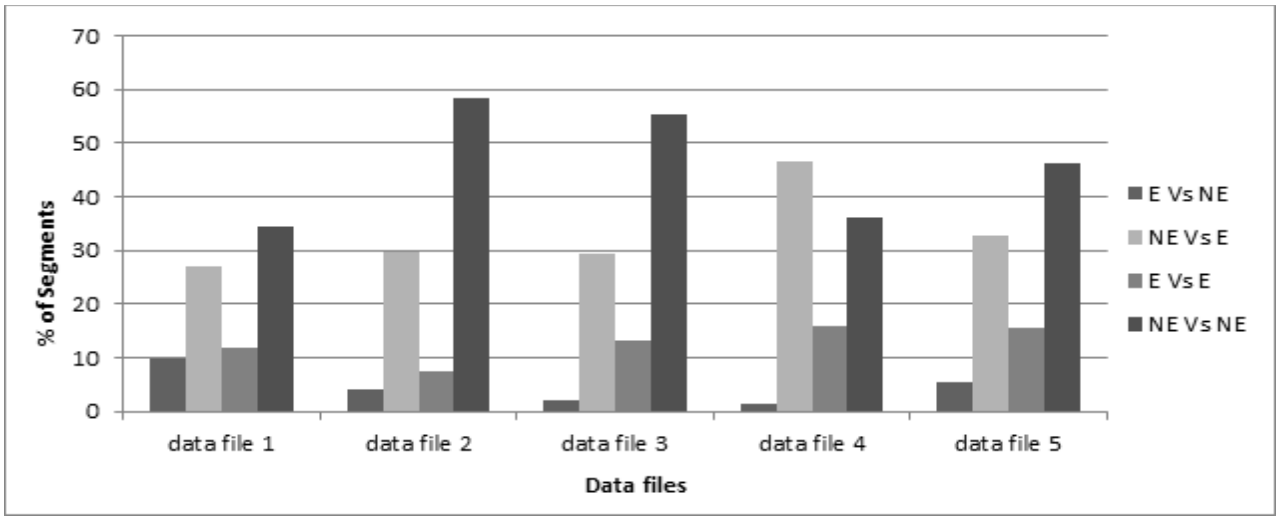

Figure 12. Percent of segments of each type (experiment-3).

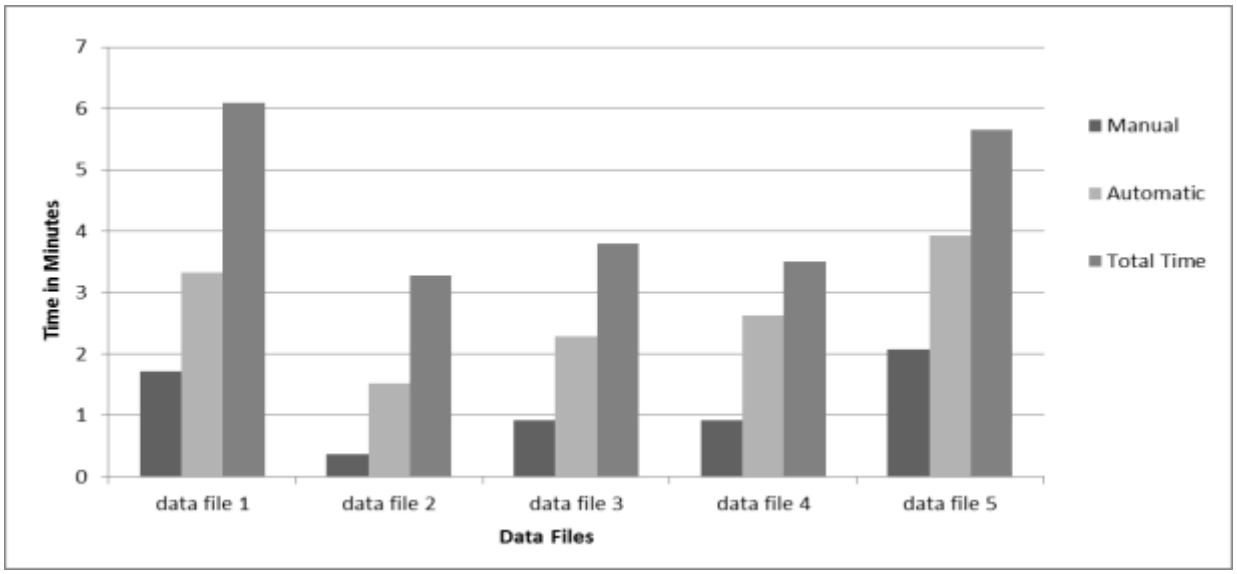

Figure 13. Total time of segments classified as excessive (Experiment-3).

and all the feature values have the same percentage of type-I and type-II errors.

Figure 15 shows the total time of segments classified as excessive by the software program and the manual process for the three features.

The "light dark" bars in Figure 15 represent the total video time recorded by the eye tracker. Manual classification of the video file, depicted by the dark bars, shows 0.92 minutes of excessive effort. The automatic classification of the video file for all three features shows 1.95 minutes of excessive effort time, which is depicted by the light bars in Figure 15. All the features have an acceptable value of type-II errors at 8.57%.

The entire set of experiments including all the data files is detailed in [11].

VII. RESULT EVALUATION

In this section, we evaluate and discuss the results of the experiments conducted in this work. Our criteria for success are based on 1) the number of type-II errors and 2) the minimal time to investigate the usability issues with an acceptable level of type-II errors. Based on discussions with several engineers in the company sponsoring this work and other companies, we are assuming that 15% of error of type-II is the upper bound for being considered as acceptable. This is also consistent with a two-step approach where after a first pinpoint analysis stage, which allows for high rate of errors but provides significant reduction in evaluation time, the errors identified are fixed, leading to a more rigorous pinpoint analysis with lower error bound. The results are evaluated based on the performance of each pattern recognition method on individual features. In addition, the

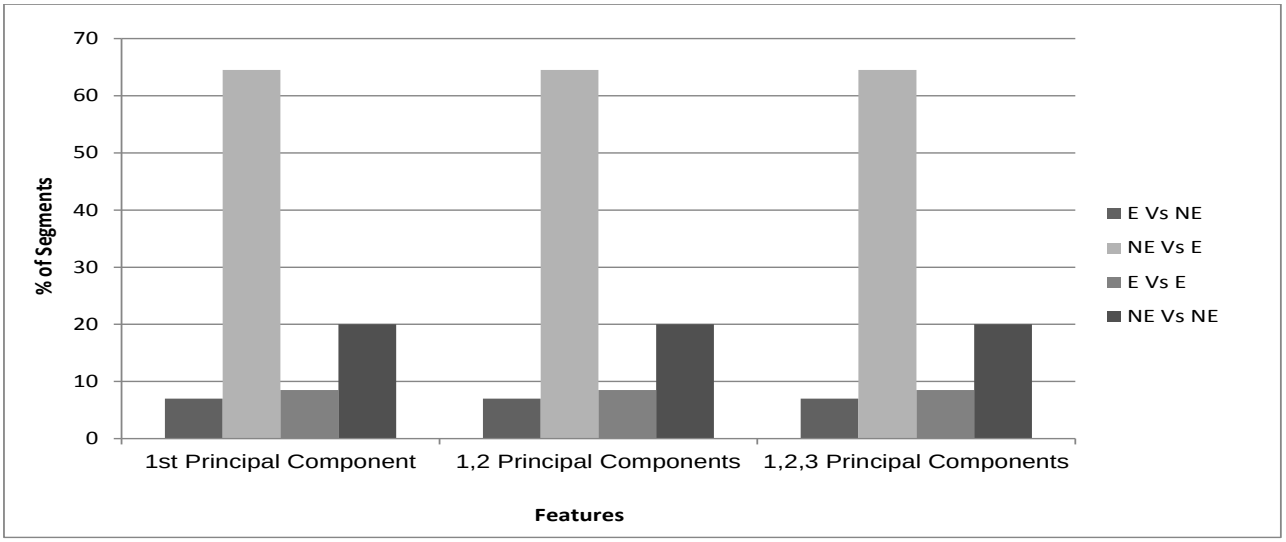

Figure 14. Graph of percentage of segments of each type (file 3, experiment-4).

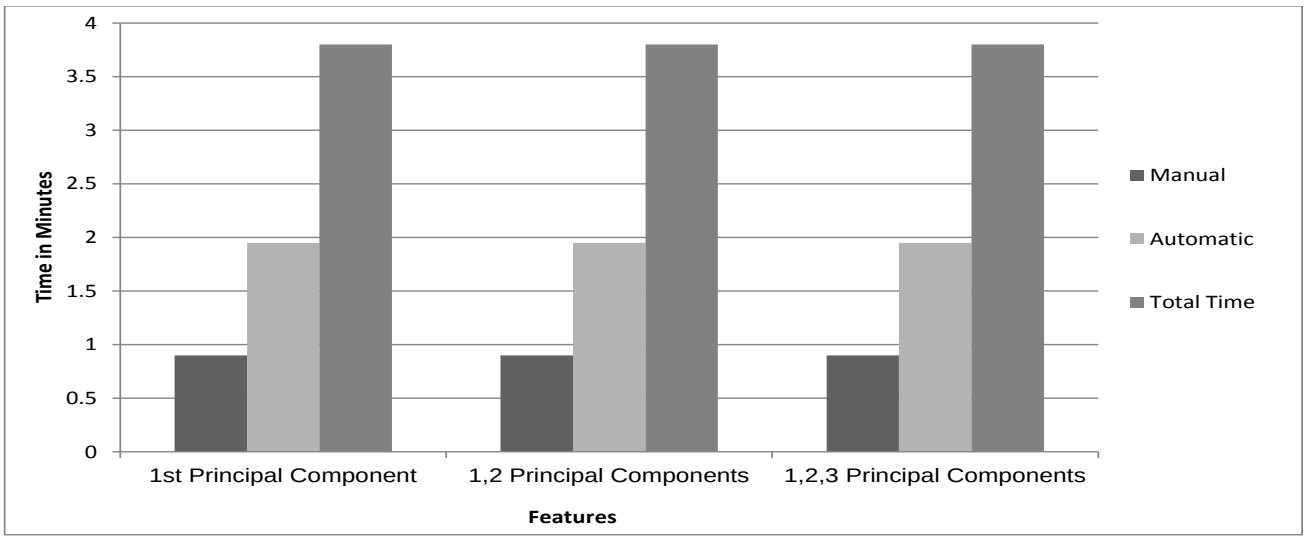

Figure 15. Total time of excessive effort segments (file 3, experiment-4).

overall performance of each pattern recognition method is evaluated.

Tables I to IV summarize the results of the experiments. An additional set of tables, which contains the entire results, can be found in [11].

A. Applying the threshold method

The following observations are derived from Table I:

1. The results of Table I show that the threshold method on the feature value, *number of fixations*, gives good results in terms of type-II errors but, the average inspection time is relatively high when compared to other feature values. The average value of type-II errors for the *number of fixations* is 3.3%. The *average saccade amplitude* and the *average eye path traversed* follow the *number of fixations* in terms of type-II errors.
2. A threshold on the *average fixation duration* performs well in terms of minimal inspection time with an acceptable value of 9.8% for type-II errors.
3. A feature value with minimum number of total errors is *average eye path traversed*. This feature value is a good choice when inspection time is not a crucial factor.
4. The inspection time is not completely correlated to type-I errors. In the case of *average fixation duration*, the inspection time is 1.67 minutes with 29.4% of type-I errors. On the other hand, the *average saccade amplitude* with almost the same percentage of type-I errors has higher inspection time than *average fixation duration*.
5. The values of the average number of *excessive effort segments* for all features are in close proximity to each other. However, the percentage of type-I and type-II errors differs invariably. Indicating that the segments classified as excessive are different for each feature value.
6. Despite the fact that the percentages of total errors for each feature value are in close proximity to each other, the inspection time varies. This delineates that the segments classified as excessive are different for each feature value.

B. Applying heuristic feature selection and K-means clustering.

The following observations are derived from Table II:

1. The results of Table II show that the K-means clustering on the feature subset - *number of fixations, number of saccades, average eye path traversed, average fixation duration, and average saccade amplitude*, gives good results in terms of type-II errors with an average value of 5.4%.

TABLE I. AVERAGE VALUES OF EXPERIMENT-1 RESULTS.

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type- I errors	avg. % type- II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
# Fix	17.2	95	28.4	3.3	31.7	2.7	62.1
Avg. Fix Dur.	18.2	95	29.5	9.9	39.4	1.6	37.4
#Sacc	32	95	21.8	10.5	32.2	2.9	64.1
Sacc Amp.	17.6	95	29.1	4.6	33.7	2.5	56.4
Eye Path	17.8	95	25.7	5.1	30.8	2.6	57.7

TABLE II . AVERAGE VALUES OF EXPERIMENT-2 RESULTS

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type -I errors	avg. % type -II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
#fix	29.1	95	27.2	6.6	33.9	2.4	56.2
#sacc	23.5	95	17.8	8.9	26.7	2.0	45.1
eye path	19.7	95	18.0	10.1	28.1	1.6	37.5
#fix, #sacc, eye path	23.2	95	18.3	8.6	26.9	1.9	44.5
#fix, #sacc, eye path, avg. fix dur., avg. sacc amp.	29.2	95	32.6	5.4	38.0	2.5	56.3

However, the average inspection time is relatively high when compared to other feature values. The *number of fixations* follows the above identified feature value in terms of type-II errors.

2. Clustering using the *average eye path traversed* performs well in terms of minimal inspection time with an acceptable value of 10.1% for type-II errors.
3. A feature value with minimum number of total errors is the *number of fixations*. This feature value is a good choice when the inspection time is not a crucial factor.
4. The average number of *excessive effort segments* for the *number of fixations* and the feature subset with the following features: *number of saccades*, *average eye path traversed*, *average fixation duration*, *average saccade amplitude* are the same. However, the inspection times vary. Indicating that the segments classified as excessive are different for each feature value.
5. Unlike the results of the threshold method, the percentages of total errors for each feature value vary by a wide margin when applying the K-means clustering on different feature subsets.

C. Using PCA

The results summarized in Table III are compared with the results obtained from Experiment-1 to compare the performance of the threshold method on the *first principal component* with the performance of thresholding on all the other features including the *number of fixations*, the *average fixation duration*, etc. Experiment-1 result evaluation shows that the feature value, *number of fixations*, gives good results in terms of type-II errors. The average percentage of type-II errors for the *number of fixations* is 3.3%, whereas it is 4.1% for the *first principal component*. Initially, the *average saccade amplitude* and the *average eye path traversed* succeeded the *number of fixations* in terms of performance. However, the new results place the threshold on the *first principal component* right after the *number of fixations* with respect to type-II errors.

The inspection times for the first principal component and for the *average fixation duration* are 2.7 and 1.6 minutes, respectively. A threshold on the *average fixation duration* performs better than the *first principal component* in terms of lower inspection time and an acceptable 9.8% for type-II errors.

D. Applying K-means clustering on principal components.

Table IV shows the average values of all the features used in Experiment-4 over the five data files. The average type-II error is very high when using the K-means on the principal components. The average inspection time is only 1.96%. When taking type-II errors also into consideration, this method is not suitable to identify *excessive effort segments*.

Of all the pattern recognition methods used, a threshold on *number of fixations* yields the best results in terms of type-II errors with a reduction of more than 40% in manual inspection time and is followed by a threshold on the *first principal component*. The K-means clustering on the feature subset with the features: 1) *number of fixations*, 2) *number of saccades*, 3) *average saccade amplitude*, 4) *average fixation duration*, and 5) *average eye path traversed* ranks third.

The K-means clustering on the *number of saccades* yields the best results and precedes the threshold method on *average fixation duration* in performance.

.

TABLE III. AVERAGE VALUES OF EXPERIMENT-3 RESULTS

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type -I errors	avg. % type- II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
1st principal components	16.6	95	27.5	4.1	31.6	2.7	61.2

TABLE IV. AVERAGE VALUES OF EXPERIMENT-4 RESULTS

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type- I errors	avg. % type- II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
1st, 2nd & 3rd principal components	28.6	95	24.4	12.6	37.0	2.0	43.6

.

VIII. CONCLUSIONS AND FUTURE RESEARCH

The framework presented in this research enables software developers to efficiently identify usability issues and deficiencies in numerous types of applications thereby optimizing the time spent on software-usability testing and validation.

Excessive effort segments, which typically relate to usability issues, are identified by applying pattern recognition techniques, such as K-means clustering algorithm, thresholding, PCA, and feature selection. The analysis of the experiments conducted in this paper shows that the time taken for software usability testing can be reduced by 40% or more.

In this research, the time between two consecutive keyboard/mouse clicks by a user is considered as a segment and serves as the basic pattern for the pattern recognition techniques. Equal time slicing of user's software interaction session can be used instead and the performance results can be analyzed and compared with the results from this research.

Further refinement of pattern recognition techniques can be pursued to minimize errors and inspection time. Also, more focus can be placed on the criteria for manual classification of video segments thus allowing *excessive effort segments* to be identified more accurately in the first place.

Another direction for future research is to automate some of the manual steps in this process. This can include software that automatically logs the data from users' interaction session, manipulates the data, and without human intervention, identifies the *excessive effort segments*. This can significantly reduce time taken for the usability testing.

In this work, we have concentrated on pattern recognition techniques that do not rely on human intelligence. Hence, the results are generated using non-supervised learning procedures. A surrogate approach can use supervised learning procedures. This involves conducting experiments using training data sets to manually arrive at an archetype that can be applied on any data set to generate the output.

Finally, we plan to investigate the utility of dynamic UI, which adapts to the user experience. For example, widget placement might change based on usage patterns. Pinpoint analysis is expected to be a crucial tool for evaluating the effectiveness of the dynamic interface approach for identifying related deficiencies.

ACKNOWLEDGMENT

This research was funded in part by Emerson Process Management [24], an Emerson business.

REFERENCES

[1] D. K. V. Dasari, D. E. Tamir, O. V. Komogortsev, G. R. LaKomski, and Carl J. Mueller, "Pinpoint analysis of software usability," ICCGI 2013, The Eighth International Multi-Conference on Computing in the Global Information Technology, Nice, France - July, 2013, pp. 66-71.

[2] J. S. Dumas and J. C. Redish, "A Practical Guide to Usability Testing," OR, USA, Intellect Books., 1999.

[3] J. Nielsen, Usability Engineering, San Francisco, Boston, Academic Press, 1993.

[4] J. Rubin and D. Chisnell, Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests, Indianapolis, Wiley Publishing, Inc., 2008.

[5] D. E. Tamir, C. J. Mueller, O. V. Komogortsev, "A learning-based framework for evaluating software usability," *the ARPN Journal of Systems and Software*, June 2013, pp. 65-77.

[6] C. J. Mueller, D. E. Tamir, O. C. Komogortsev, and L. Feldman, "Using designer's effort for user interface evaluation," *IEEE International Conference on Systems, Man, and Cybernetics*, Texas, USA, October 11, 2009, pp. 480-485.

[7] ISO/IEC 9126-1: 2001, Software Engineering-Product Quality, Part-1, Quality Model, Geneva, Switzerland: International Standards Organization, 2001.

[8] ISO/IEC 9126-1: 2001, Software Engineering-Product Quality, Part-2, External Metrics, Geneva, Switzerland: International Standards Organization, 2001.

[9] D. E. Tamir, O. V. Komogortsev, and C. J. Mueller. "An effort and time based measure of usability," 6th Workshop on Software Quality, 30th International Conference on Software Engineering, Leipzig, Germany, 2008, pp. 35-41

[10] D. E. Tamir, et al. "Detection of software usability deficiencies," International Conference on Human Computer Interaction, FL, 2011, pp. 528-536.

[11] D. K. V. Dasari, Pinpoint analysis of software usability, Thesis Report, Texas State University, Computer Science, December 2012.

[12] C. Holland, O. V. Komogortsev, D. Tamir. "Identifying usability issues via algorithmic detection of excessive visual search," Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI), Austin, TX, 2012, pp. 1-10.

[13] A. Poole and L. J. Ball, Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future Prospects, Encyclopedia of Human Computer Interaction: Idea Group, 2004.

[14] M. A. Just and P. A. Carpenter, "Eye fixation and cognitive processes," Cognitive Psychology, vol. 8, 1976, pp. 441-480.

[15] J. T. Tou and R. C. Gonzalez, Pattern Recognition Principles, Reading, MA: Addison-Wesley Publishing, Inc., 1974.

[16] R. O. Duda, P. E. Hart, and D. G. Stock, Pattern Classification, 2nd Ed., Indianapolis, Willey International, 2001.

[17] D. E. Tamir and A. Kandel, "The pyramid fuzzy c-means algorithm," International Journal of Computational Intelligence in Control, 2 (2), 2012 pp. 65-77.

[18] H. Ebbinghaus, Memory: A Contribution to Experimental

Psychology, 1885, http://psychclassics.yorku.ca/Ebbinghaus/memory3.htm, retrieved June 2014.

[19] Anonymous, "MATLAB Product Help," MATLAB, 2013, http://www.mathworks.com/help/, retrieved June 2014.

[20] E. T. Hvannberg and C. L. Lai, "Classification of usability problems (CUP) scheme," Nordic conference on Human-computer Interaction, Oslo, Norway, 2006, pp. 655-662.

[21] N. Nakamichi, S. Makoto, and S. Kazuyuki, "detecting low usability web pages using quantitative data of users' behavior," Proceedings of the 28th international conference on Software engineering, New York, NY, 2006, pp. 569-576.

[22] S. Makoto, N. Noboru, H. Jian, S. Kazuyuki, and N. Nakamichi. "Webtracer: A new integrated environment for web usability testing," 10th Int'l Conference on Human - Computer Interaction. Crete, Greece, June 2003, pp. 289-290.

[23] Anonymous, "Tobii X60 & X120 Eye Trackers: User Manual," Tobii, 2013, http://www.tobii.com/Global/Analysis/Downloads/User_Manuals_and_Guides/Tobii_X60_X120_UserManual.pdf, retrieved June 2014.

[24] Anonymous, "Emerson Process Management," Emerson, 2014, http://www.emersonprocess.com, retrieved June 2014.