

An Improved Decision Support System for Software Evaluation Using Weighted Sum Technique

Benjamin, Ubokobong Effiong

Department of Computer Science, Akwa Ibom State Polytechnic, Ikot Osurua, Ikot Ekpene, Nigeria

ABSTRACT

This research study focused on an improved decision support system for software evaluation using weighted sum. Given the increase in the number of software packages in the market today, there is a challenge of knowing which is most effective to solve the problems of an organization without creating problems. It is in view of the need to select the most appropriate software among alternatives that gave rise to the concept of software evaluation. Software evaluation is the process of assessing the quality of different software systems so as to choose the most productive one. Evaluating and selecting software packages that meet an organization's requirements is a difficult software engineering process. Selection of a wrong software package can turn out to be costly and adversely affect business processes. One of the most common techniques used for evaluating software components is weighted sum. In this technique, the criteria for evaluating the software are clearly defined and values are assigned as weight to each criterion. The total of the weight placed on each criterion reveals the level of effectiveness of the software component. For the scope of this study, the evaluation is carried out under three different criteria categories which are; vendor, hardware/software requirements and cost/benefits.

KEYWORDS: Improved, Decision Support System, Software Evaluation, and Weighted Sum Technique

1. Introduction

A successful evaluation is not simply picking a product based on intuition. It involves a formal process, the right mixture of evaluators, and a specific quantifiable set of evaluation criteria. The process should include how to handle differences in scoring by the evaluators. The task of choosing a software component for a specific function in order to integrate it in a software system is a typical case of multi-criteria decision making that frequently occurs in Software Engineering. Consider a decision maker with a set of components to fulfill a function in a software system, for example creating digital signatures on files. A number of decision factors will come into play such as functional suitability, security, performance efficiency, interoperability and costs. Some of these may pose conflicts: For example, increased security may come at the price of decreased performance efficiency or increased price. The decision maker has to follow a trustworthy and repeatable procedure to choose the component that best fulfills the objectives at hand (Becker et al, 2013). The domain of component selection presents an interesting case of multiple criteria decision support systems (MCDSS) since it exhibits a number of peculiarities:

1. A comparably large number of decisions of a very similar kind is made.

2. The number of alternatives and decision criteria can be quite large.
3. The decision criteria are rather well understood in terms of the facets and quality aspects that are evaluated.

Software can be evaluated with respect to different aspects, for example, functionality, reliability, usability, efficiency, maintainability, portability. In earlier times evaluation of software took place at the end of the developing phase, using experimental designs and statistical analysis, evaluation is nowadays used as a tool for information gathering within iterative design: "Explicit human-factors evaluations of early interactive systems (when they were done at all) were poorly integrated with development and therefore ineffective. They tended to be done too late for any substantial changes to the system still be feasible and, in common with other human-factors contributions to development, they were often unfavourably received.

Decision Supports Systems (DSS) are computer-based information systems designed in such a way that help managers to select one of the many alternative solutions to a problem. It is possible to automate some of the decision-making processes in a large, computer-based DSS which is sophisticated and analyze huge amount of information fast. It helps corporate to increase market share, reduce costs, increase profitability and enhance quality. The nature of problem itself plays the main role in a process of decision making.

1.1. Statement of the Problem

As institutions and organizations spend huge amount on Enterprise resource planning (ERP) packages and other computer software that cost hundreds of thousands and even millions of dollars, purchasing a software solution is a high expenditure activity that consumes a significant portion of companies' capital budgets. Selecting the right solution is an exhausting process for companies. Therefore, selecting a software package that meets the requirements needs a full examination of many conflicting factors and it is a difficult task. Most times the software bought do not meet the needs of the institution or organization despite the huge amount. To avoid the problem of software ineffectiveness, this has led researchers to investigate better ways of evaluating and selecting software packages.

1.2. Aim and Objectives of the Study

The aim of the study is to develop an improved decision support system for software evaluation that will help organizations to determine the effectiveness of a software product based on its features and capabilities. The following are the objectives of the study:

1. To design a decision support system for software evaluation using quantitative method for software evaluation and selection.
2. To develop a software that will assess the software features to determine their level of effectiveness.
3. To compare a system that will maintain record of software evaluation records.

1.3. Scope/Significance of the Study

This study covers advanced decision support system for software evaluation using weighted sum. It is limited to the capturing of the weighted sum of software features and the determination of the best software option based on the total weight of its features. Evaluation is based on three different criteria categories which are: The vendor, hardware/software requirements and cost/benefits of the software system.

However, the significance of the study is that it will help institutions and organizations evaluate the effectiveness of a software product. The study will also serve as a useful reference material to other researchers seeking for information concerning the subject.

2. LITERATURE REVIEW

2.1. Software Selection using Decision Support Systems

According to Bandor (2006), when performing purchase analysis and selecting a product as part of a software acquisition strategy, most organizations will consider primarily the requirements (the ability of the product to meet the need) and the cost. The method used for the analysis and selection activities can range from the use of basic intuition to counting the number of requirements fulfilled, or something in between. The selection and evaluation of the product must be done in a consistent, quantifiable manner to be effective. By using a formal method, it is possible to mix very different criteria into a cohesive decision; the justification for the selection decision is not just based on technical, intuitive, or political factors. Decision making is considered one of the most critical activities done in organizations. To support this complex process for individuals, a variety of independent, standalone information systems called Decision Support Systems (DSSs) have been developed in the two last decades.

2.2. Decision Support System Overview

Decision support systems (DSS) emerged in the 1970. It is defined as a computer-based system designed to actively interact with an individual decision maker in order to assist him to make better decisions based on information obtained. The decision process is broadly defined as a bundle of correlated tasks that include: gathering, interpreting and exchanging information; creating and identifying scenarios, choosing among alternatives, and implementing and monitoring a choice. Briefly, the decision process refers to some techniques or processing rules aiming at structuring the context, timing or content of communication. DSS was designed to solve ill or non-structured decision problems. Problems where priorities, judgements, intuitions and experience of the decision maker are essential, where the sequence of operations such as searching for a solution, formalization and structuring of problem is not beforehand known, when criteria for the decision making are numerous, in conflict or hard dependent on the perception of the user and where resolution must be acquired at restricted time (Bhargadav and Power, 2015).

2.3. Multi-Criteria Decision Support System (MCDSS) for Software Component Selection

According to Becker et al (2013), numerous approaches have been proposed for the general problem of software component evaluation and selection. Most methods for component selection employ a variation of the standard five steps:

1. Define criteria
2. Search for components
3. Shortlist candidates
4. Evaluate candidates
5. Analyze results and choose component

Frequently employed approaches for evaluating and selecting components include the usage of simple scoring and weighted sum approaches, the Analytic Hierarchy Process (AHP), or iterative filtering. Others use methods based on utility analysis to tackle the incommensurability of decision factors. In particular in cases of strict requirements on trustworthiness and reliable selection of components, evidence-based decisions using controlled testing are recommended (Becker and Rauber, 2010). For the scenario of component selection, using goal-based requirements modeling and utility analysis is especially suitable for a number of reasons: The decision models strongly build on quality attributes that lend themselves to requirements engineering approaches; the anomaly of rank reversal should be avoided; and the number of analytical steps that for example the application of the AHP requires is in many cases prohibitive. Still, the problematic aspect of all approaches for component selection that can be considered trustworthy, i.e. evidence-based and formalized, is the high complexity and effort involved in creating suitable evidence. This begins with the unambiguous specification of criteria for quality attributes, which can be quite challenging, and extends to the evaluation of components, i.e. the process of assigning values to decision criteria.

2.4. Software Evaluation Techniques

Software evaluation is multi-criteria decision making problem that refers to making preference decisions over the available alternatives. At this point, the various software evaluation techniques are discussed and their strengths and weaknesses are examined.

1. Analytic Hierarchy Process (AHP) Software Evaluation Technique

AHP has been widely used for evaluation of the software packages. AHP has been identified as an important approach to multi-criteria decision-making problems of choice and prioritization. AHP is based on a hierarchical framework of criteria. The upper level deals with the goal of the selection process. The next level defines the major factors which are subdivided into their constituents in lower levels of hierarchy. The bottom level contains the alternatives to be analyzed.

Strengths of Analytical Hierarchy Process (AHP):

1. AHP enables decision makers to structure a decision making problem into a hierarchy, helping them to understand and simplify the problem.
2. It is flexible and powerful tool for handling both qualitative and quantitative multi-criteria problems.
3. AHP procedures are applicable to individual and group decision making.

Weaknesses of Analytical Hierarchy Process (AHP):

1. AHP is time consuming because of the mathematical calculations and number of pair-wise comparisons that increases as the number of alternatives and criteria increases.
2. The decision makers need to re-evaluate alternatives when the number of criteria or alternatives are changed.
3. Ranking of alternatives depends on the alternatives considered for evaluation hence adding or deleting alternatives can lead to changes in the final rank.

2. Feature Analysis Software Evaluation Technique

Feature analysis technique for software evaluation uses different software feature criteria to evaluate the software such as:

Process Related Criteria

1. Development lifecycle: What development lifecycle best describes the methodology (e.g. waterfall)?
2. Coverage of the lifecycle: What phases of the lifecycle are covered by the methodology (e.g. analysis, design, and implementation)?
3. Development approach: What development approach is supported (i.e. top-down or bottom-up)?
4. Application domain: Is the methodology applicable to a specific or multiple application domains?
5. Scope of effort: What size of MAS is the methodology suited for (i.e. small, medium, or large)?
6. Agent nature: Does the methodology support only homogeneous agents, or heterogeneous agents?
7. Support for verification and validation: Does the methodology contain rules to allow for the verification and validation of correctness of developed models and specifications?

Strength of Feature Analysis Software Evaluation Technique:

1. Evaluation can be done to any required level of detail by organizing evaluation in different ways such as

screening mode, case study, formal experiment and survey.

2. It is used not only for technical evaluation but also for evaluation of viability of supplier.

Weakness of Feature Analysis Software Evaluation Technique:

1. Producing the single number from the individual scores may be misleading because many different combinations of numbers can produce the same aggregate score.

3. Weighted Average Sum (WAS) Software Evaluation Technique

Another technique used for evaluation of software package is the weighted scoring method. In this method weights and rating scales are assigned to each criterion. The weight reflects the relative importance of each of the criteria while the rating scale indicates how easily each package is able to meet the specific criterion. The rating scales are then multiplied by weight factor of each criterion. Using this scheme a score is calculated for every criterion for each tool. These scores are then totaled to produce a score for each criteria category and the average is also computed. Finally, the categorical scores are compared to determine the highest.

There are many different methods for deriving risk values, but descriptions of these methods are out of scope for this report. Additional references on risk can be found in the bibliography. Regardless of which risk calculation method you choose to follow, it is important to keep in mind that the scoring mechanism presented above is based on a "higher is better" score, and most risk calculations are based on a "lower is better" score. The two methods should be used individually and not combined into a single score for evaluation purposes. Table 2.1 shows an example legend for scoring when using weighted average sum evaluation technique.

Table 2.1: Example legend for scoring requirements

Score Value	Definition
1.0	Alternative fully satisfies business requirement or decision criterion.
0.5	Alternative partially satisfies business requirement or decision criterion.
0.0	Unknown or null/balanced (The alternative neither satisfies nor dissatisfies business requirement or decision criterion.)
-0.5	Alternative partially dissatisfies business requirement or decision criterion.
-1.0	Alternative fully dissatisfies business requirement or decision criterion [Litke 02].

Source: Bandor (2006)

Strengths of Weighted Average Sum Software Evaluation Technique:

1. Main advantage of WAS is its ease of use.

Weaknesses of Weighted Average Sum Software Evaluation Technique:

1. Weights to the attribute are assigned arbitrary and it is very difficult to assign weight when number of criteria is high.
2. To obtain a score using this method a common numerical scaling is required.
3. Difficulties emerge when WAS is applied to multi-dimensional MCDM problems.

2.5. Weighting Sum Software Product Selection Factors

Before selecting specific products, institutions should consider each of the factors or criteria for evaluation, balancing as far as possible the merits of specific products against the general features of the system. The selection of a specific product requires attention to:

1. Software reliability.
2. Availability of technical support by the institution to the users.
3. Availability of support by the software supplier to the institution and the users.
4. Cost to the institution (i.e., local server support).

A quantification approach, using weighted average mean can be used for the software product evaluation. The evaluation responses may be weighted using points scoring criteria and scorecards. Results can then be compared quantitatively according to the evaluation totals and average. The review and the analysis of the responses are recommended to be performed in the following sequence.

1. Analyze each evaluation response using a 'score card'.
2. Review each requirement listed in the score card and check the answer(s). It is recommended to use a simple 'Yes or No' marking, or a combined weighting and scoring method to indicate to what degree the score card requirements are met by the evaluator.
3. Repeat the process, using a new scorecard for each software product.
4. The evaluation criteria have to formalize the requirements towards the software products.

Table 2.2: Matrix weighted evaluation score card

SCORECARD	Reviewer: Requirements	Date: 8/1/06	Products: A	Software qualification
Evaluation Criteria	Weighting	Evaluation Score	Points	Notes
Workflow functionality	3 x	3	9	good
Purchase order	3 x	3	9	strong
cost	3 x	3	9	excellent
Needs for training	3 x	3	9	limited
TOTAL SCORE				
MAX SCORE				

Source: Stoilova and Stoilov, (2005)

By using a defined and understood set of discrete values, the subjectivity of the evaluation is significantly reduced. The raw values can be based on the information shown in Table 2.3. There are only five values used, ranging from 1.0 to -1.0 in increments of 0.5. Note the use of negative values and the effects on the scoring. Instead of just assigning a value of 0, the use of negative values permits the application of a "penalty" value where not meeting the criterion would be detrimental.

Table 2.3: Scoring legend for software criteria evaluation

Score Value	Definition
1.0	Alternative fully satisfies business requirement or decision criteria
0.5	Alternative partially satisfies business requirement or decision criteria
0.0	Unknown or null/balanced (The alternative neither satisfies nor dissatisfies business requirement or decision criterion)
-0.5	Alternative partially dissatisfies requirement or decision criterion
-1.0	Alternative fully dissatisfies requirement or decision criterion

2.6. Software Evaluation Models

Balsamo et al (2006) in contrast to software evaluation techniques, software evaluation models determine the frame of the evaluation, which consists of:

1. Choosing techniques appropriate for the life cycle, and
2. Setting the focus with respect to the objects under study and the measurement criteria.

Evaluation models may provide a standardized treatment of establishing (potentially) successful procedures in the practice of evaluation, and are a necessary tool for a comparing different types of software evaluation. Any descriptive evaluation procedure must be combined with some kind of predictive technique to result in an applicable evaluation model; furthermore, some preparatory steps are necessary. For example, the evaluation model, which consists of the IsoMetric questionnaire as a basic technique, standardizes the preparatory steps "choosing the tasks" and "choosing the user group(s)"; it also standardizes the preparation of the report by an expert, and the structure of a "usability review", which consists of a standardized result presentation and a Walkthrough technique. There are three classes of evaluation models:

Method Driven Models: These models offer a frame for software evaluation based on a collection of techniques. The models are only applicable if the evaluation procedures fits perfectly the problems encountered by the user and the system developers.

Criteria Driven Models: More or less abstract criteria are defined and refined; the evaluation in these models aims at a measurement of the criteria.

Method Driven Evaluation Models: The centre of method driven evaluation models consists of the arrangement of evaluation techniques, amended by the regulation of preparatory and subsequent tasks. A method driven evaluation model can be perceived as a complex evaluation technique as well. An example is EVADIS (Evaluation of Dialogue Systems II), which is well tested for office automation software. The EVADIS II model combines interviews, simplified task analysis, and expert judgement in the following steps:

1. Installation and exploration of the software.
2. Analysis and relevance weightings of the tasks, construction of test tasks.
3. Analysis of the user characteristics. Selection of relevant ergonomic test items.
4. Evaluation of the software, based on the results of the first three steps.
5. Interpretation of the results and composing a test report.

The first three preparatory steps can be handled in parallel; they result in testing tasks and a list of ranked ergonomic evaluation criteria, mainly based on the principles of ISO 9241 (Part 10). The ranks of the criteria are deduced from the user profile, and they determine the test items which are chosen from an item database. In step four, the testing tasks are evaluated by an expert who steps through the tasks, and answers the questions formulated in the ergonomic items. The recorded answers form the basis for the test report. Every step of EVADIS II is supported by a large amount of supporting material such as databases and guidelines, which allows a domain expert with only a small amount of knowledge of software evaluation to form a well-founded opinion on the topic.

2.7. Empirical Related Work

Koziolok (2009) surveyed the state-of-the-art in research of performance evaluation methods for component-based software systems. The survey classified the approaches according to the expressiveness of their performance modelling language and critically evaluated the benefits and drawbacks. The area of performance evaluations for component-based software engineering has significantly matured over the last decade. Several issues have been understood as good engineering practice and should influence the creation of new approaches. A mixed approach, where individual components as well as the deployment platform are measured and the application architecture and the usage profile are modelled, is advantageous to deal with the complexity of the deployment platform while at the same time enabling early life-cycle performance predictions. The necessary parameterized performance modeling language for software components has become more clear. Including benchmarking results for component connectors and middleware features into application models using model completions exploits the benefits of model-driven development for performance evaluation.

The survey conducted by (Koziolok, 2009) benefits both researchers and practitioners. Researchers can orient themselves with the proposed classification scheme and assess new approaches in the future.

They can select methods according to their specific situation and thus increase the technology transfer from research to practice.

Babar et al (2015), conducted a study on a framework for classifying and comparing software architecture evaluation methods. The researchers opined that different software engineering communities have developed different techniques for characterizing their respective quality attributes and the methods to evaluate software systems with respect to that particular quality attribute, e.g., real-time, reliability, and performance. These assessment techniques study a specific quality attribute in isolation. In reality, however, quality attributes interact with each other. For example, there is generally a conflict between configurability and performance; performance also impacts modifiability, availability affects safety, security conflicts with usability, and each quality attribute impacts cost. That is why it is important to find an appropriate balance of quality attributes in order to develop a successful product. One of the most significant features of method differentiation and classification is the number of quality attributes a method deals with.

The software architecture (SA) evaluation methods specifically studied by (Babar et al, 2015) are: Scenario-based Architecture Analysis Method (SAAM), Architecture Tradeoff Analysis Method (ATAM), Active Reviews for Intermediate Design (ARID), SAAM for Evolution and Reusability (SAAMER), Architecture-Level Modifiability Analysis (ALMA), Architecture-Level Prediction of Software Maintenance (ALPSM), Scenario-Based Architecture Reengineering (SBAR), SAAM for Complex Scenarios (SAAMCS), and integrating SAAM in domain-Centric and Reuse-based development (ISAAMCR). These are scenario-based methods, a category of evaluation methods considered quite mature. There are also some attribute model-based methods and quantitative models for SA evaluation, but these methods are still being validated and are considered complementary techniques to scenario-based methods. There is, however, little consensus on the technical and non-technical issues that a method should fully address and which of the existing methods is most suitable for a particular issue. There is not much work done on systematic classification and comparison of the existing methods. Moreover, there is not much guidance on the desirable features of the evaluation methods and their usefulness.

Software Architecture (SA) evaluation can be performed for a number of purposes, e.g., risk assessment, maintenance cost prediction, architecture comparison, trade-off analysis and so forth. No one method can be considered as equally good for all types of assessment objectives as different methods are optimized to achieve different evaluation goals. The common goal of most of the evaluation methods is to evaluate the potential of the designed architecture to facilitate or inhibit the achievement of the required quality attributes. For example, some architectural styles, e.g., layered architectures, are less suitable for performance sensitive systems, even though they usually result in highly flexible and maintainable systems. In order to achieve maximum benefit from an assessment activity, the goals of the evaluation need to be explicitly defined. The goals of assessment help software architect make a number of critical decisions with regard to selection of a specific method and deliverable required.

There is at least one common goal found in all surveyed methods, which is prediction-based assessment of the quality of a system at the architecture level. However, each method has a specific view and different approach to achieve the goal: SAAM and its variants (specifically SAAMCS and ISAAMCR) are mainly geared to identify the potential architectural risks. However, SAAMCS focuses on exposing the boundaries of SA with respect to flexibility using complex scenarios, while, ISAAMCR integrates SAAM in domain-centric reusable development process; SAAMER evaluates the designed SA for evolution and reusability and provide a framework for SA analysis; ALMA specializes in predicting one quality attribute (i.e., modifiability) and there are three possible objectives to be pursued: risk assessment, maintenance cost prediction, and SA comparison; ARID performs suitability analysis of intermediate design artifacts; ATAM identifies and analyses sensitivity and trade-off points as these can prevent the achievement of a desired quality attribute.

Bandor (2006) also conducted a research on Quantitative Methods for Software Selection and Evaluation. The author was of the opinion that when performing a “buy” analysis and selecting a product as part of a software acquisition strategy, most organizations will consider primarily the requirements (the ability of the product to meet the need) and the cost. The method used for the analysis and selection activities can range from the use of basic intuition to counting the number of requirements fulfilled, or something in between. The selection and evaluation of the product must be done in a consistent, quantifiable manner to be effective. By using a formal method, it is possible to mix very different criteria into a cohesive decision; the justification for the selection decision is not just based on technical, intuitive, or political factors. The report describes various methods for selecting candidate commercial off-the-shelf packages for further evaluation, possible methods for evaluation, and other factors besides requirements to be considered. It also describes the use of a decision analysis spreadsheet as one possible tool for use in the evaluation process.

In addition, Koziolk, (2009) carried out a study on Performance Evaluation of Component-based Software Systems: A Survey. He believed that Performance prediction and measurement approaches for component-based software systems help software architects to evaluate their systems based on component performance specifications created by component developers. Integrating classical performance models such as queuing networks, stochastic Petri nets, or stochastic process algebras, these approaches additionally exploit benefits of component-based software engineering, such as reuse and division of work.

3. System Analysis and Design

3.1. Research Methodology

The data used for the development of the research was gotten from the internet, textbooks and articles. The contributions of other researchers on the subject were examined so as to gather relevant information. Questionnaire forms were also issued to experienced users of the software to be evaluated in order to obtain raw data to ascertain their effectiveness.

The system analysis and design methodology used to analyze the system is Object Oriented Analysis and Design Methodology (OOADM). OOADM applies object orientation in the analysis and design as a software engineering approach that models a system as a group of interacting objects. Object oriented analysis and design is the analysis and design of a system from the object point of view.

3.2. System Analysis

System analysis has to do with examining a system in order to understand its step by step operations so as to identify its benefits and areas of limitation that require improvements. Analysis of the existing and proposed system is examined at this point.

3.2.1. Analysis of the Existing System

In the existing system of software evaluation using weighted sum, it is manually carried out on a sheet known as an evaluation score card. Fig 3.1 below, gives an illustration of how software evaluation is manually done using a score card.

Architecture of the Existing System

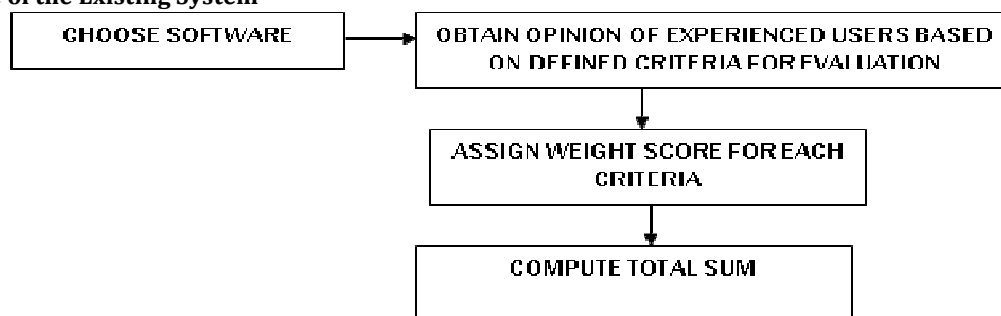


Fig 3.1: Architecture of existing system

In the architecture of the existing system shown above, the software to be available is chosen, the opinion of users of the software is obtained based on the defined criteria. Weighted score values are assigned to the defined criteria and the total weight sum is computed.

3.2.2. Analysis of the Proposed System

The proposed system is such that it will provide a grading or evaluation interface that will enable users carry out software evaluation of different software that are of the same application category. The software will be evaluated side by side. The system will also provide expert system remark on the level of reliability after assessment. The evaluation will be done based on three different criteria which are:

- Vendor evaluation

- Hardware/software evaluation
- Cost/benefits evaluation.

Each criterion for software evaluation outline above has sub-criteria that will be assigned evaluation values ranging from 2 to -2. At the end, the total sum is computed and the percentage sum is also computed. Expert system remark is also provided indicating which software is better. The evaluation key values and their meaning are shown in figure 3.2 below:

EVALUATION KEY	
VERY SATISFIED	2
SATISFIED	1
NEUTRAL	0
DISSATISFIED	-1
VERY DISSATISFIED	-2

Fig 3.2: Evaluation key values

Architecture of the Proposed System

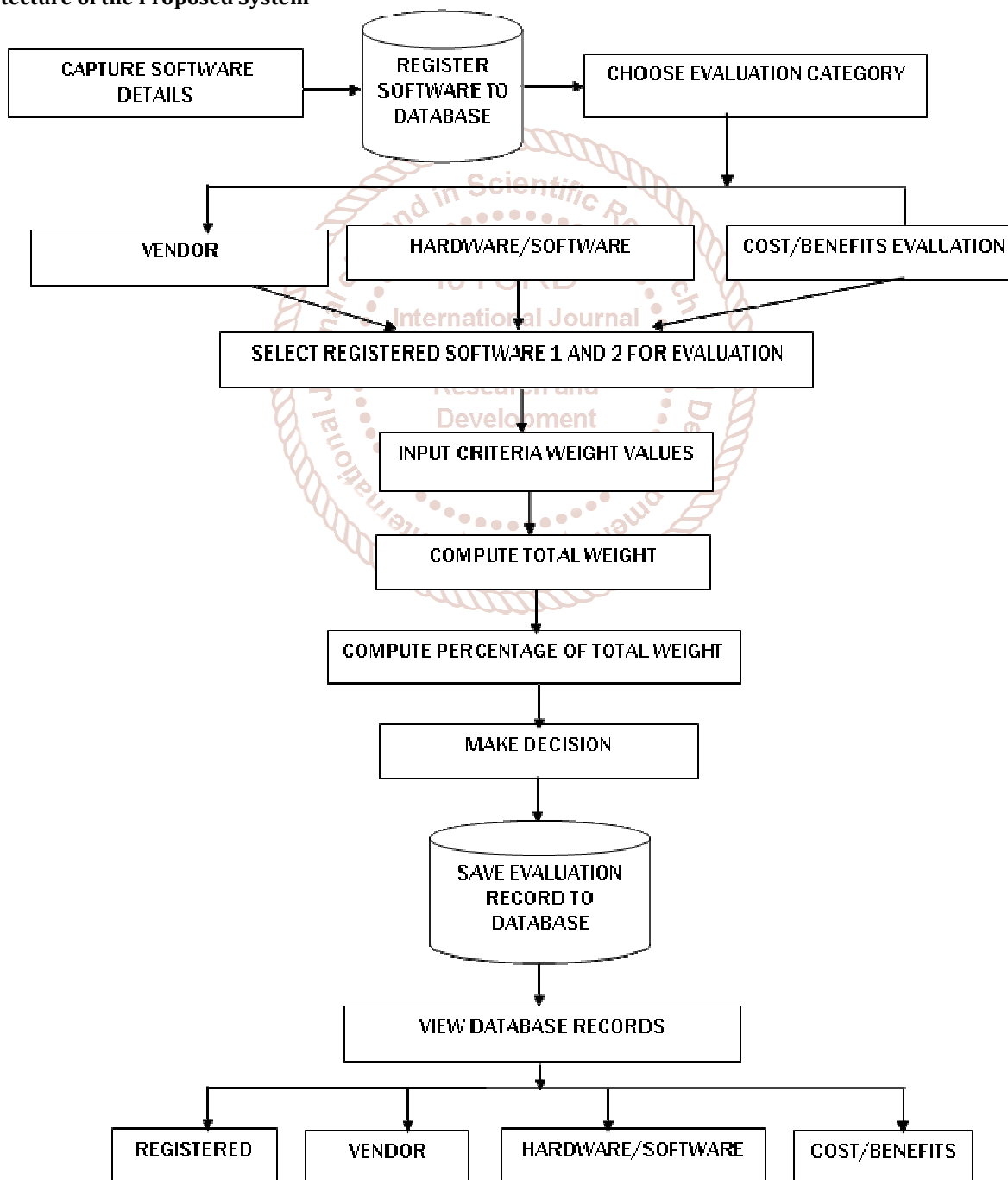
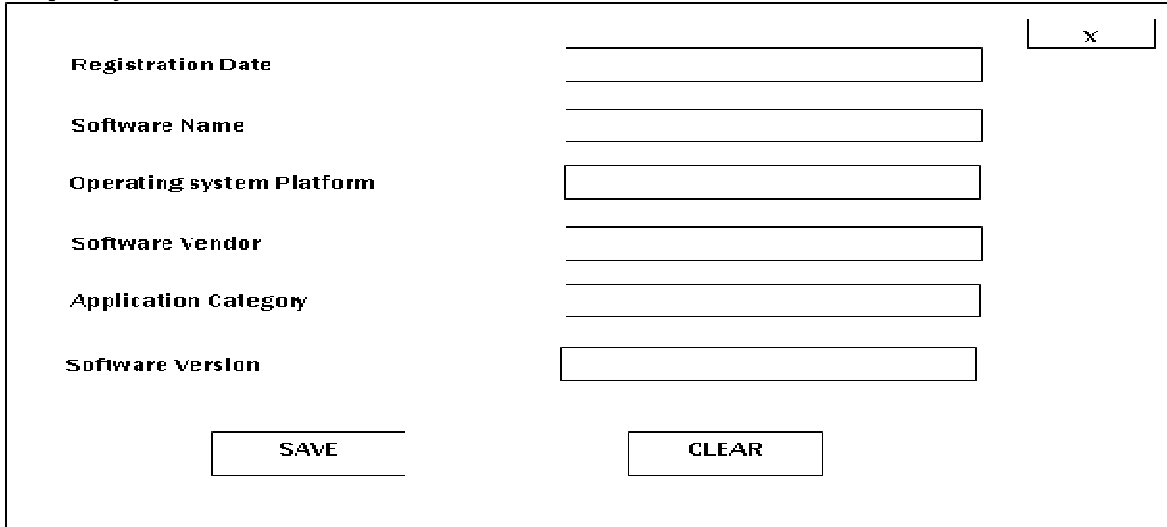


Fig 3.3: Architecture of proposed system

3.3. System Design

The system design has to do with the layout of the system and it comprises of the input and output layout and Algorithm design and program flow chart.

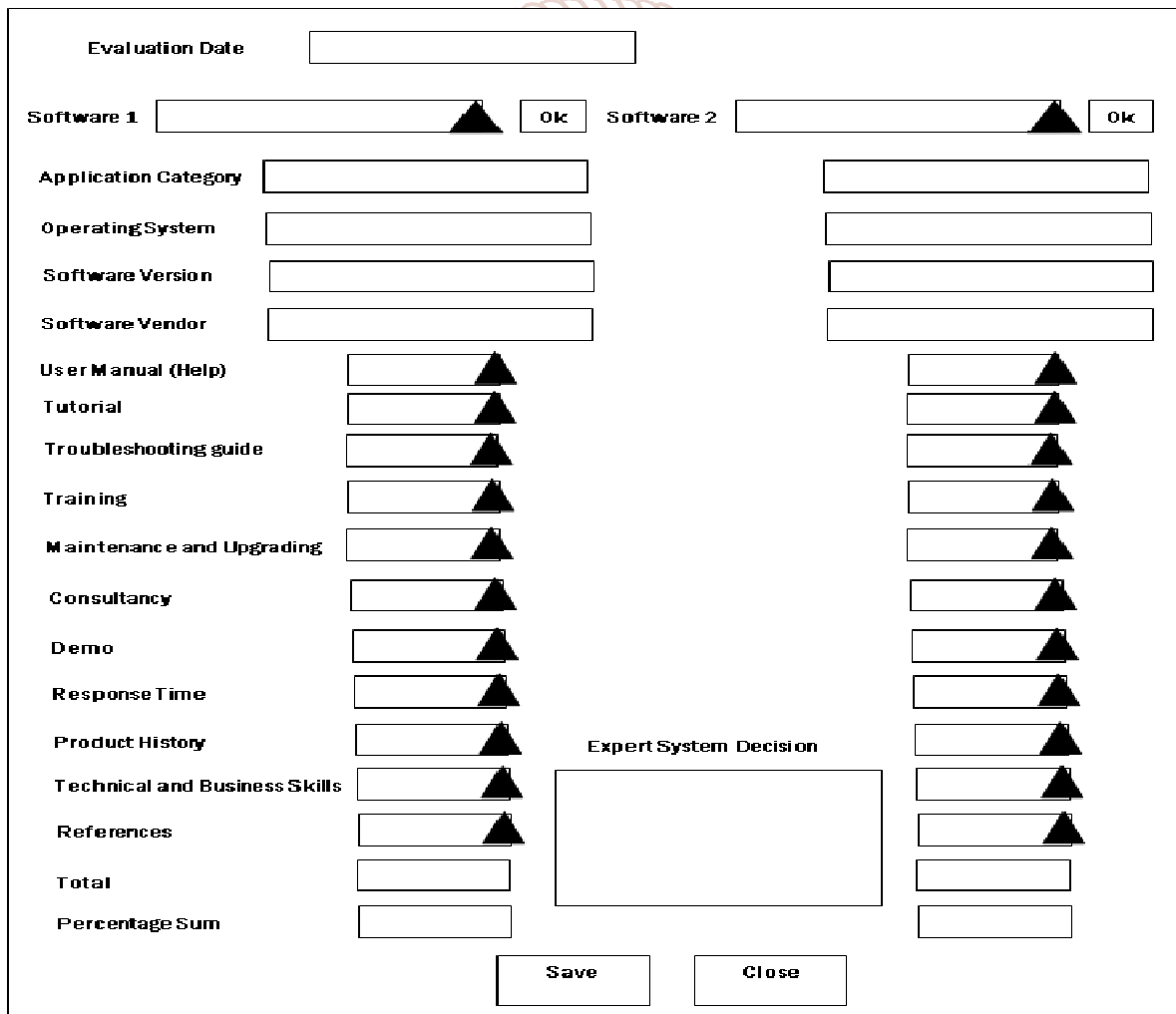
3.3.1. Input layout



The form for software registration includes the following fields and controls:

- Registration Date**: Input field
- Software Name**: Input field
- Operating system Platform**: Input field
- Software Vendor**: Input field
- Application Category**: Input field
- Software Version**: Input field
- SAVE**: Button
- CLEAR**: Button
- X**: Close button

Fig 3.4: Software registration input layout



The vendor evaluation form is structured as follows:

- Evaluation Date**: Input field
- Software 1**: Input field with **Ok** button
- Software 2**: Input field with **Ok** button
- Application Category**: Input field
- Operating System**: Input field
- Software Version**: Input field
- Software Vendor**: Input field
- User Manual (Help)**: Input field with up arrow
- Tutorial**: Input field with up arrow
- Troubleshooting guide**: Input field with up arrow
- Training**: Input field with up arrow
- Maintenance and Upgrading**: Input field with up arrow
- Consultancy**: Input field with up arrow
- Demo**: Input field with up arrow
- Response Time**: Input field with up arrow
- Product History**: Input field with up arrow
- Technical and Business Skills**: Input field with up arrow
- References**: Input field with up arrow
- Total**: Input field
- Percentage Sum**: Input field
- Expert System Decision**: Large text area
- Save**: Button
- Close**: Button

Fig 3.5: Vendor Evaluation input layout

Fig 3.6: Hardware Software Evaluation input layout

3.3.2. Output Layout

See Appendix B

3.3.3. Algorithm

Step 1: Start

Step 2: Login

Step 3: If login is success goto step 4 else goto step 2

Step 4: Display main menu

Step 5: Input choice

Step 6: If choice is software registration goto step 7 else goto step 8

Step 7: Input registration details and save to database.

Step 8: If choice is reliability evaluation goto step 9 else goto step 12

Step 9: Input evaluation category

Step 10: If evaluation category is vendor goto step 11

Step 11: Select software 1 and 2 and Input vendor evaluation details

Step 12: Compute total weight

Step 13: Compute Percentage of total weight

Step 14: Display expert system decision

Step 15: If evaluation category is hardware/software goto step 16

Step 16: Select software 1 and 2 and Input hardware/software evaluation details

Step 17: Compute total weight

Step 18: Compute Percentage of total weight

Step 19: Display expert system decision

Step 20: If evaluation category is cost/benefits goto step 21

Step 21: Select software 1 and 2 and Input cost/benefits evaluation details

Step 22: Compute total weight

Step 23: Compute Percentage of total weight

Step 24: Display expert system decision

- Step 25: If choice is database records goto step 26
 Step 26: Input choice
 Step 27: If choice is registered goto step 28
 Step 28: Display database of registered software
 Step 29: If choice is vendor evaluation goto step 30
 Step 30: Display database of vendor evaluation records
 Step 31: If choice is hardware/software evaluation goto step 32
 Step 32: Display database of hardware/software evaluation records
 Step 33: If choice is cost/benefits evaluation goto step 34
 Step 34: Display database of cost/benefits evaluation record.
 Step 35: If choice is quit goto step 36
 Step 36: Stop

4. System Implementation and Documentation

4.1. System Design Diagram

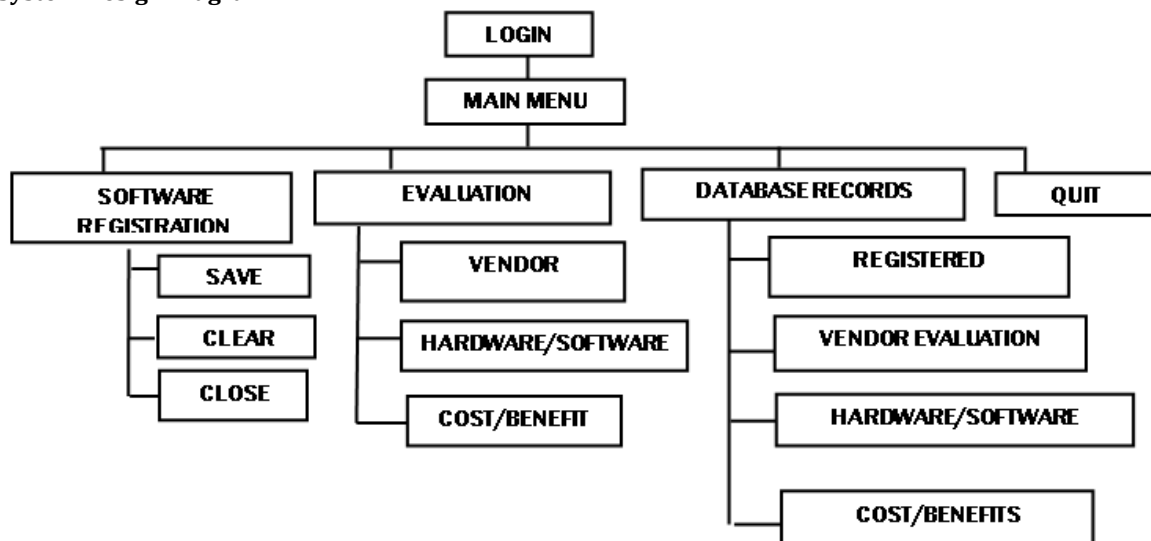


Figure 4.1: System Design Block Diagram

4.2. Choice of Programming Language

The programming language used is Visual BASIC.NET. The language was chosen because it enables the creation of applications with a graphical user interface, containing controls such as text fields, combo box, labels, command buttons etc.

4.3. Analysis of Modules

The system is made up of four main modules as shown in the system flow diagram. They are;

Software Registration: This module enables the registration of software for evaluation

Evaluation: This module aids the evaluation of registered software. It is made up of three sub-modules namely: vendor, hardware/software, cost/benefits. These sub-modules are the criteria for evaluating the software, so as to arrive at a decision.

Database Records: This module aids the user to view the database records of registered software, vendor evaluation records, hardware/software evaluation records and cost/benefits evaluation records.

Quit: This module terminates the program

4.4. Programming Environment

The programming environment used for the development of the application is windows 7 operating system and the integrated development environment (IDE) chosen for the development of the system is Visual BASIC 6.0.

The hardware and software requirements for successful implementation of the system are stated at this point.

The hardware requirements are;

- Pentium iv computer system
- Super video graphic array monitor
- 1 GB RAM
- Keyboard
- Mouse
- Uninterruptible power supply (UPS)

The software requirements are:

- Microsoft Visual Basic 8.0 (Visual Basic.NET)
- Microsoft Access 2003

4.5. Implementation

Implementation is the process of replacing the old system with the new system. There are four different ways of replacing the old system with the new system. The reasons for choosing one implementation type over another depend upon; how quickly must the changeover happen? How important is it to prevent data loss? What will the cost of the changeover be?

Phased implementation: Takes longer to complete the implementation but the risks to the business are less than for direct changeover. The new system can be split into separate working parts, part of the old system is replaced with the new one until the replaced part is working properly. Continue the process until the entire old system has been replaced by the new system.

Direct changeover: In this system the old system is no longer available and everything must run on the new system. Problems with the new system can cause major problems for the business, only suitable for non-critical systems.

Parallel Running: Highly fault tolerant, new system and the old system are used with extra staffs recruited to run the new system but it is very expensive. Both systems continue to run until the new system is working properly then the old one is discarded.

Pilot Running: If the business has many different offices or sites then this is an option. One single site is chosen and the old system is replaced with the new system in the same way as direct changeover but only on one site, the rest of the business continue to use the old system. Once the new system is shown to work well in that one 'pilot' site then the new system can replace the old one in the rest of the company.

The system implementation method recommended and chosen by the system developer is the parallel running so as to prevent data loss.

5. Conclusion and Recommendations

5.1. Conclusion

From the foregoing, it can be seen that a successful evaluation is not simply picking a product based on intuition. It involves a formal process, the right mixture of evaluators, and a specific quantifiable set of evaluation criteria. The process should include how to handle differences in scoring by the evaluators. Defining the evaluation technique used for the evaluation is very important.

Weighted sum evaluation technique can be adopted to easily ascertain the effectiveness of software based on defined criteria. From the foregoing, it can be seen that, requirements drive selection criteria, careful consideration must be given to the identification of selection criteria, pilots and demonstrations are essential selection tools, product and technology maturity must be considered. By systematically analyzing co-occurrence, correlation and impact of decision criteria across cases, it should be possible to integrate recommender systems into the decision making workflow that can provide increased guidance and warn decision makers of potential risks and opportunities based on others' experiences.

5.2. Recommendations

The following recommendations are offered based on the study:

- More research should be conducted on decision support system for software evaluation/assessment.

- Software development companies should conduct survey programs to assess the reliability of their software product.
- Raw data used for the assessment should be obtained from users of the software system.
- The criteria utilized in making decision on the effectiveness of a software should be expanded to include more aspects so as to improve the reliability information.
- Proper testing and debugging should be done before software systems are published or placed in the market.
- Trial versions of software are important for users to assess the effectiveness of the software.

REFERENCES

- [1] Babar, M. A., Liming Zhu, Jeffery, R. (2015). *A Framework for Classifying and Comparing Software Architecture Evaluation Methods*. National ICT Australia Ltd. and University of New South Wales, Australia.
- [2] Balsamo, S., DiMarco, A. Inverardi, P. and Simeoni, M. (2004) Model-based performance prediction in software development: A survey. *IEEE Trans. Softw. Eng.*, 30(5):295-310, May 2004.
- [3] Bandor, Michael S. (2006). *Quantitative Methods for Software Selection and Evaluation* Copyright 2006 Carnegie Mellon University Technical Note CMU/SEI-2006-TN-026 September 2006
- [4] Becker, C. and Rauber, (2010) A. Improving component selection and monitoring with controlled experimentation and automated measurements. *Information and Software Technology*, 52, 641-655
- [5] Becker, C. Kraxner, M., Plangg, M., Vienna, A. (2013). Improving decision support for software component selection through systematic cross-referencing and analysis of multiple decision criteria. University of Technology. 2013 46th *Hawaii International Conference on System Sciences*
- [6] Bhargava, H. and Power, D. (2015). *Decision support systems and web technologies: A status report*.
- [7] Koziolok, H. (2009). *Performance Evaluation of Component-based Software Systems: A Survey*. ABB Corporate Research, Industrial Software Systems, Wallstadter Str. 59, 68526 Ladenburg, Germany
- [8] Stoilova, K., Stoilov, T. (2005). *Software Evaluation Approach*. European Commission, project №FP6-027178 VISP, and National Scientific Fund of Bulgaria, project № ВУ-МИ-108/2005