**British Journal of Mathematics & Computer Science**
**9(4): 328-356, 2015, Article no.BJMCS.2015.207**
***ISSN: 2231-0851***

**SCIENCEDOMAIN** *international*
www.sciencedomain.org

SCIENCEDOMAIN

# Software Quality Improvement Using Force-field Analysis

**Makanjuola Daniel**[1*]

[1]*Computer Science Department, Salem University, Lokoja, Kogi State, Nigeria.*

***Article Information***

DOI: 10.9734/BJMCS/2015/14714
Editor(s):
(1) Qiang Duan, Information Sciences & Technology Department, The Pennsylvania State University, USA.
(2) Ke-Lin Du, Chief Scientist, Enjoyor Labs, Hangzhou, China, Department of Electrical and Computer Engineering, Concordia University, Canada.
(3) Sun-Yuan Hsieh, Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan.
Reviewers:
(1) Anonymous, The University of the West Indies, Jamaica.
(2) Anonymous, California State University, USA.
(3) Anonymous, University of Pennsylvania, Indiana, USA.
(4) Anonymous, Universidade do Algarve, Portugal.
Complete Peer review History: http://www.sciencedomain.org/review-history.php?iid=1145&id=6&aid=9524

***Empirical Brief Research Article***

*Received: 15 October 2014*
*Accepted: 09 April 2015*
*Published: 01 June 2015*

## Abstract

Today, global demands for quality products and services have moved quality assurance to the forefront. Just as quality is a determining factor for the survival of a product in a competitive market, quality system will be a requirement for the competitive survival of Information Technology (IT) firms in the future.
In my plight to improve the performance of software systems, I began by seeking the opinions of software developers, having conducted personal interview with some of the developers; It was perceived that some of the software developers were being exaggerative about the processes of software development. This made me to change my target audience from software developers to the users of software systems. The reason for this is because it is the software users that are always at the receiving end of every lapses, failure, error and mistakes incurred in the cause of developing software systems.
From my experience in this research, it was observed that, there are two categories of users; the power users who made use of software systems to render services and the common users who are the receiving audience of the services rendered by the power users.
The information received from the common users apparently reflects the level of performance of the software systems, thus I decided to work with the information received from them.
In the plight of improving the current level of performance of the software systems, I perceived that the best way to accomplish this is by shedding away the wrong practices and pattern of software development process by the developers, as well as the unethical ways of using the software systems (by the software users) and also to promote the supporting factors that influence the performance of software systems.
Using the requirement negotiation process embedded in win-win spiral model of software development, and with the aid of force field analysis, I have been able to establish a bench-mark that forms a foundational model for every software development project.

*Corresponding author: danielmakanjuola@salemuniversity.edu.ng;

*Keywords: Software quality; force field analysis; software improvement.*

## 1 Introduction

For a few golden moments after the inception of the water-fall model [1], it appeared that the software field had found a sequence of common anchor points around which people could plan, organize, monitor and control their projects. These anchor points functioned as the milestones that enabled companies, government, organizations and standard groups to establish a set of interlocking regulations, specifications and standards that covered a full set of software project needs, such as cost and schedule estimation, project plans, reviews and audits, configuration management, and quality assurance. Extensive completion criteria were established for each milestone, such as completeness, consistency, traceability, testability and feasibility. As the usage of software product progresses, it appeared that, water-fall software development model could not meet users' immediate needs, hence the emergence of Evolutionary software development model which appears to be faster than waterfall model in terms of "time-to-market" schedule with the aid of rapid prototyping but the short fall of this model was later identified as most software systems produced are always poorly structured. Later, the use of Formal methods was produced which was soon criticized to be too expensive to adopt.

Boehm [2] incorporated risk analysis into software development model which was helpful for a while, but soon enough researchers have come to realize that the problems with software products is not with the development models, but with the issue of software quality produced. Hence, there is an emergent need and interest of ensuring the quality of the software produced is good enough to make it perform efficiently and optimally in every task it is designed for.

## 2 Background of Study

In her article published in 1994 [3], Capper Jones reported that poor quality was one of the most common reasons for schedule overruns. She also reported that quality is implicated in close to half of all canceled projects.

June Verner (2006) [4] of the National Information and Communication Institute of Australia, when carrying out a risk management campaign on software development noted that most developers believed they would achieve success in their respective projects because of one or more of the following reasons:

i. They had a sense they had delivered a quality product
ii. They had a sense of achievement
iii. They had enough independence to work creatively
iv. The requirements were met and the system worked as intended at the point of installation.

However, the research conducted revealed the following significant information:

i. 33% of projects said they had no risk but 62% of those failed
ii. 60% of organizations have no process to measure benefits and risk associated with a software projects
iii. 86% of projects had a business case, but 60% ignored it
iv. 5% of projects had no Project Manager and 16% change Project Manager at least once (and that was correlated to project failures).

Verner concluded in her report that, very few organizations use risk management in their software development project and those that do rarely manage those risks.

It is therefore of great significance to develop a concept that will reveal the value of each software projects in (in terms of cost and benefits) as well as the risk involved in each software project, because success comes from a culture that investigates and deals with problems.

Hence, there is an emerging need for software quality improvement methodology that checkmates the basic activities in each phase of the software development. This makes modified version of spiral model (win-win) sufficiently good enough for the analysis of risks involved in the software development process.

## 3 Motivation of Research

Software quality improvement has been one of the major concerns in the software industries. The issues of software quality improvement boil down to the stage of requirement identification, software requirement gathering and specification, risk analysis and management, and system delivery.

The prolific measures of the above-mentioned factors could only be realized by making intensive analysis of the supporting and restraining factors that influence the improvement of any software product.

Force-field [5] analysis is a Physics concept that set up a stage to show-case the supporting and restraining forces against a proposed project. It is an important concept mostly used by project managers for building an understanding of the forces that will drive or resist a proposed change. Force field analysis helps to weigh the importance of the factors relating to a particular decision and to know whether a plan is worth implementing. It is a specialized method of weighing pros and cons of a proposed project.

## 4 Statement of Problems

Many software developing institutes and researchers have been able to incorporate a number of alternative process models: risk-driven (spiral), reuse-driven, legacy-driven, demonstration-driven, design to cost or schedule, incremental and hybrid combination of these with the waterfall or evolutionary development models.

Also, they have been able to use a risk-driven software development model [6] to explicitly enhance good feasibility and the risk involved in each phase (loop) of the software development process. Furthermore, spiral models have been modified to win-win spiral model to identify the win-win factors, win-lose factors and the lose-lose factors.

Egbokhare [7] also came out with series of analysis to identify the roles, perceptions, interactions and experiences in software development process. In her publication, she was able to identify the problems and give suggestive clues on how to take care of some of the factors contributing to a poor performance or failure in software systems, especially in the area of human resources which is the fundamental structural elements for any "people intensive" activity such as software development.

All of these authors, developers and engineers have contributed immensely to providing quality software products interms of showing us a supportive and restraining factors to providing a quality software product, but they have not been able to give clues on how to trade-off those restraining factors by providing reinforcing concepts that could be help reduce the possibility of those restraining or demoting factors.

## 5 Aim and Objectives

The aim of this research is to study the present state of software development in Nigeria with special focus on the process involved in developing software systems. Our area of special concentration will be channeled toward identifying the following vital factors:

i. Identifying the helping or supportive factors in the software development process.
ii. Identifying the restraining or demoting factors in the software development process.
iii. Reduce the strength of the forces opposing the development of quality software systems.
iv. Increase the forces pushing or supporting the development of quality software systems.

The above mentioned objectives are to be worked on, in order to improve the quality of the overall software products.

## 6 Significance of Study

Every software process improvement effort begins with a process assessment, which involves an honest introspection and careful analysis of an organization's software development process to identifying factors contributing to a successful software development project and also the constraints and restraining factors that lead to a deficient software product.

At the end of the research, I was able to come out with a quality development model adaptable by various software industries and research institutes. This will go a long way in reducing or perhaps eliminating the demoting factors contributing to poor quality software products. Also the model promises to promote the supporting factors by consistently applying a process improvement framework that could be used to continuously improve on the existing software development techniques.

## 7 Scope of Study

This research identified the base practices adopted in software development process. The significant factors affecting development of software systems were looked into. Also, the research paves way for the analysis of supporting factors and constraints involved in the software development processes. The research was mainly carried out to determine the supporting factors that should be strengthen and the constraints necessary to be reduced, in order to produce a quality and reliable software product.

## 8 Limitation of Research

In the cause of carrying out this field research, few establishments deprived us from accessing their software systems (e.g Central Bank of Nigeria, PHCN, and UAC Foods). The reason was tagged to some securities and logistics issues. This actually deprived me from accessing the set of software systems that most people criticize and complaint about.

Also, many software users had their software systems developed from outside the country (e.g Banks, Capital Trust Stock Exchange, Starcomms, TMS Travels etc.) This also deprived me from reaching their software developers to actually study and access the software development processes from their respective developers.

## 9 Methodology of Research

Today's global demand for quality products and services has moved quality assurance to the forefront. Just as quality is a requirement in the modern market-place, quality systems will be a requirement for the competitive survival of corporations and organization in the future. To have quality systems, companies need to develop specific plans for gathering software requirement, designing software, writing programs and compilers and linking the software package together to form a whole system.

Quality assurance in software development is becoming more and more synonymous with ensuring the flow of critical information in methodology observed, management techniques and technical approaches. To incorporate good quality methodology into the software development process in developing country like Nigeria, we need to:

i. Know the current level of performance of the existing software systems used in Nigeria
ii. Identifying the pro's and con's in each software development project

iii. Identifying the actions involved in software development process and the resulting reactions or consequences associated to each action on every software development project
iv. To compare the ideal situations and realties in each software development process.
v. Attempt to bridge the gap between the user's priority and developer's views.

To achieve the above-listed goals, structured questionnaires and key informant interviews were used to elicit necessary data from project managers, software developers and users of software systems. The organizations visited include:

i. Software Development Organization
ii. Educational Institutions
iii. Banks
iv. Research Institutes
v. State Government Parastatals

The heterogeneous sample indulged in this research consists of the following:

i. Software team leaders of Software Development Organizations
ii. Software Developers
iii. Users of Software Systems

## 10 Survey Instruments

The two main survey instruments used for this research area as listed below:

i. Questionnaires
ii. Key Informant Interview

### 10.1 Questionnaires

180 structured questionnaires containing questions related to some selected software quality attributes were administered to users of software systems. For the purpose of this study, we grouped users into two main categories:

i. Power Users
ii. Common or End Users

#### 10.1.1 Power users

These are the personnel who make use of the software systems to render services to the populates.

#### 10.1.2 Common or end users

These are the receiving audience who make use of the services being rendered by the power users.

The categorization became necessary because the people in different categories share different views about the software systems they use. Also, the common users mostly do not understand the technical terms and the functionality of various parts of the system they use. Although they can sense if their software systems are having problems but they do not know how to explain exactly what is wrong with their systems. Power users sometimes take part in the technicality and maintenance of the software systems but common users do not. The main aim of the questionnaire was to identify the current level of performance of the existing software systems in Nigeria. To enhance the content validity, the questionnaires were presented to some IT experts and the suggestions made were effected before the final copies were distributed. Five States in Nigeria where there is high level of software usage were randomly selected for this study.

Firstly, to affirm the reasons of this research, we started with the common users. About 180 structured questionnaires on the software qualities of our interest (about 14 of them) were administered, of which 150 were usable for the analysis. The first part of the questionnaires sought information about demography of the users in order to know their level of relevance with IT tools. The second part sought information about the type (and purpose) of the software systems while the third part sought opinions from the respondents on their software quality assessment. The following sections constitute the analysis of the questionnaires.

## 10.2 Demographic Information

Table 1 to Table 5 reflects the demographic profile of the common users. Table 1 enumerates the academic profile of the common users interviewed; the information obtained from the Table 1 reflects that only 1 personnel which constitutes just 0.7% has a primary school certificates, about common users (52%) possessed Senior Secondary School Certificates while 71 of our common users have minimum of Bachelor Degree Certificates (47.3%).

Also Table 2 reflects the gender information about the common users. There are about 120 Males which constitutes about (68.0%), as well as 48 Females (32.0%).

Table 3 gives account of several age-bracket within our common users. About 25 people (16.7%) fall within 18-20 years. Also, 104 users covered 21-30 (about 69.3%), 18 people (12%) constitutes 31-40 year age bracket, and finally, 3 common users which constitutes (2.0%) fall within 41-50 years.

Furthermore, Table 4 gives account of the relevant job experiences of our common users. About 51 people (37%) acquired between 1-5 years job experiences, 78 people equivalent of (52%) of our common users claimed between 6-10 years, while up to 28 users (14%) claimed they have acquired above 10 years job experience.

Finally, Table 5 recorded the employment status of the common users interviewed, in this segment; about 30 common users which constitute (20.0%) are employers of labour while 120 common users (about 80%) are employees.

The main focus of this analysis is to seek the opinions of the common users on the qualities of the software systems enumerated. Table 6 represents a five-point scale describing the views of the common users. From this analysis, it was established that most of the software qualities are just performing at average rate. It was observed that, since the common users could not only directly influence the optimal performance of their respective software systems, they are forced to work along with whatever they are provided.

## 10.3 Necessity of Software Quality Optimization

The respondents also talked about their interest/views in optimizing each of the software qualities. Their views and responses as it is related to the necessity to optimizing each software quality are represented in the Table 7.

## 10.4 Conclusion of the First Analysis

At the end of the first analysis, the following points were extracted:

i. All respondents agree that virtually all the highlighted qualities need to be optimized
ii. Good percentage of our respondents picked “NO” as response to the highlighted qualities

iii. Even those who acknowledge excellent performance in some software qualities they are familiar with, still agree that there is a need for improvement
iv. A considerable percentage of our respondents (mostly students) always give out the work they are supposed to do on-line to somebody else because they do not understand the functionalities of the online applications.

The above conclusions and observations established a need for this research and also reflect how significant a quality software system could impact the working condition of an average user. In the plight to improving the quality of software systems at least the commonly used ones in our immediate environment, about fifteen companies making use of software systems to render different types of services to the general populates (These are categories of Power Users) were visited.

Questionnaires containing three parts were administered by the researcher to the power users who were charged with the responsibilities of developing optimal software systems for end users. Based on the demands on indigenous software systems to reach optimal performance by the common/end users initially interviewed especially with special consideration to the constraints attached to each software quality and feature in consideration, series of questions were structured into questionnaires. The set of questions in the questionnaires were channeled towards common user's view on the software systems. The reason why I had to attached top priority to users is because the developers initially interviewed are being exaggerative about their mode of development and as such, are reluctant to relay information about the weaken aspects of software development. It is believed that the users are always at the receiving end of every software failure and would be happy to accommodate prospective researchers who want to work on their software failures. The second phase of information gathering was channeled towards the power users. The questionnaires were personally distributed in order to have an informal interaction with the users who were very happy to talk about the problems they encountered while working on the software systems.

The first part of the questionnaire sought information about demographic information of the power users in order to know their level of relevance with IT tools. The second part sought information about the type (and purpose) of the software system while the third part sought the opinions of the respondents on the emphasized software qualities in their products. A questionnaire was designated to each organization. The initial aim was to carry out research on sixteen companies but eventually only fourteen of them were usable for the analysis.

Table 8 to Table 11 capture the analysis of the data gathered during my second phase of field research.

The data gathered in Table 8 confirms that 8 (57.1%) of the respondents would prefer a better software package, while 6 (42.9%) do not prefer a better software package. This implies that majority of software power users would prefer a better software package.

Table 11 information shows that 1 (7.1%) of the respondents would like to change their software developers, while 13 (92.9%) would not want to change their software developers.

**Table 1. Showing the academic profile of the examined common users**

**Academic/Professional Qualification(s)**

| | **Frequency** | **Percentage** |
|---|---|---|
| Primary | 1 | 0.7 |
| Secondary | 78 | 52.0 |
| Post-Secondary | 71 | 47.3 |
| Total | 150 | 100 |

**Table 2. Displaying the gender status of the common users**

**Sex**

| **Gender** | **Frequency** | **Percentage** |
|---|---|---|
| Male | 102 | 68.0 |
| Female | 48 | 32.0 |
| Total | 150 | 100.0 |

**Table 3. Showing the age-range of the common users**

**Age**

| | **Frequency** | **Percentage** |
|---|---|---|
| 18-20 | 25 | 16.7 |
| 21-30 | 104 | 69.3 |
| 31-40 | 18 | 12.0 |
| 41-50 | 3 | 2.0 |
| TOTAL | 150 | 100.0 |

**Table 4. Displaying the years of experience of the examined common users**

**Job Experience**

| **Years of experience** | **Frequency** | **Percentage** |
|---|---|---|
| 1-5 years | 51 | 34 |
| 6-10 | 78 | 52 |
| Above 10 years | 28 | 14 |
| Total | 150 | 100 |

**Table 5. Illustrating the job status of the common users**

**Employment Status**

| **Job status** | **Frequency** | **Percentage** |
|---|---|---|
| Employer | 30 | 20.0 |
| Employee | 120 | 80.0 |
| Total | 150 | 100 |

**Table 6. Describing the views of the common users**

| **S/N** | **Total** | **The extent of each software quality** | | | | |
|---|---|---|---|---|---|---|
| **Qualities** | | **Excellent** | **Very good** | **Average** | **Poor** | **Very poor** |
| Understandability | 150 | 9 | 22 | **94** | 9 | 16 |
| Completeness | 150 | 7 | 21 | **68** | 11 | 43 |
| Conciseness | 150 | 14 | 18 | **71** | 10 | 37 |
| Interoperability | 150 | 15 | 18 | **58** | 6 | 53 |
| Reusability | 150 | 13 | 32 | **48** | 17 | 40 |
| Integrability | 150 | 15 | 22 | **55** | 16 | 42 |
| Portability | 150 | 11 | 20 | **74** | 13 | 32 |
| Consistency | 150 | 19 | 23 | **69** | 12 | 27 |
| Maintainability | 150 | 24 | 19 | **79** | 15 | 27 |
| Testability | 150 | 9 | 14 | **69** | 30 | 28 |
| Usability | 150 | 8 | 20 | **61** | 8 | 53 |
| Reliability | 150 | 20 | 31 | **43** | 35 | 21 |
| Efficiency | 150 | 8 | 15 | **55** | 16 | 56 |
| Security/maintenance | 150 | 3 | 2 | **35** | 32 | 78 |

**Table 7. Describing the qualities that common users want to optimize**

| Qualities (S/N) | Yes, I will like to optimize it. | No, it is not necessary |
|---|---|---|
| Understandability | 132 | 18 |
| Completeness | 126 | 24 |
| Conciseness | 109 | 41 |
| Interoperability | 130 | 20 |
| Reusability | 117 | 33 |
| Integrability | 124 | 26 |
| Portability | 126 | 24 |
| Consistency | 123 | 27 |
| Maintainability | 121 | 29 |
| Testability | 123 | 27 |
| Usability | 128 | 22 |
| Reliability | 133 | 17 |
| Efficiency | 129 | 21 |
| Security/Maintenance | 138 | 12 |

## 10.5 Key Informant Interviews

The researcher personally conducted informant interviews with the power users and the IT experts in order to get details on the methods adopted in the developments of the software systems. This was ensured in order to know the lapses, weaknesses and inadequacies in the methods adopted.

This method was adopted to obtain information on user's perception of the software development process. Other questions not listed on the questionnaire for the software users but necessary to obtain a clearer picture of the state of the software performance in Nigeria were also asked.

The questions were open and close-ended with frequent probing to elaborate and clarify meaning. Responses were precise in identification of factors required. The Head of IT/ Heads of Operation of the neighboring software firms to the University metropolis were informally interviewed.

The duration of each interview was between thirty to forty-five minutes. The interviews were not rushed and the interviewees were allowed enough time to express their opinions. All interviews were conducted at the interviewees' offices. All respondents were promised protection of privacy and confidentiality.

The interviewees also gave their consent that the result can be published as long as the agreement of confidentiality and secrecy is maintained. The interview sessions demanded considerable skills which included avoiding one's own constructions into the interview, rather, allowing the interviewees to speak, determining the people to be interviewed, employing effective listening techniques and reflecting back on what a participant said, looking for opportunities to clarify meaning and evaluate the interviews.

At the end of the studies, I was able to identify the lapses, errors, mistakes, assumptions and realities taken place in the exercise of software development projects in Nigeria. Also, with the information obtained from the investigations and studies, precise analysis was made and I was able to compare the ideal situations and realities in each software development project.

Another important factor I observed is that there is always a conflict between developer's view and users' priorities. Common users share different views of software systems from the power users and also power users share different views from developers. This is the ultimate problem that this research is applied to solve (i.e bridging the gap between the user's priorities and developer's views).

**Table 8. The above table reflects the data gathered from our investigation from the power users. The table represents the level of performance of each software quality in their software systems**

**Analysis of software qualities present in the software systems used by respondents**

| ***Quality (S/N)*** | ***EX*** | ***% of Ex*** | ***V.G*** | ***% of V.G*** | ***Good*** | ***% of Good*** | ***Poor*** | ***% of Poor*** | ***Very Poor*** | ***% of Very Poor*** | ***No*** | ***% of No*** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Understandability* | *3* | *21.4* | *9* | *64.3* | *2* | *14.3* | | | | | | |
| *Completeness* | *6* | *42.9* | *3* | *21.4* | *4* | *28.6* | | | | | *1* | *7.1* |
| *Conciseness* | *6* | *42.9* | *4* | *28.6* | *2* | *14.3* | *1* | *7.1* | | | *1* | *7.1* |
| *Interoperability* | *6* | *42.9* | *4* | *28.6* | *1* | *7.1* | | | *1* | *7.1* | *2* | *14.3* |
| *Reusability* | *1* | *7.1* | *8* | *57.1* | *1* | *7.1* | | | | | *4* | *28.6* |
| *Integrability* | *7* | *50.0* | *4* | *28.6* | | | | | | | *3* | *21.4* |
| *Portability* | *8* | *57.1* | *2* | *14.3* | *2* | *14.3* | *1* | *7.1* | | | *1* | *7.1* |
| *Consistency* | *7* | *50.0* | *5* | *35.7* | | | | | | | *2* | *14.3* |
| *Maintainability* | *7* | *50.0* | *4* | *28.6* | | | | | *1* | *7.1* | *1* | *7.1* |
| *Testability* | *6* | *42.9* | *4* | *28.6* | *1* | *7.1* | | | | | *3* | *21.4* |
| *Usability* | *9* | *64.3* | *3* | *21.4* | | | *1* | *7.1* | | | *1* | *7.1* |
| *Reliability* | *5* | *35.7* | *7* | *50.0* | | | *1* | *7.1* | | | *1* | *7.1* |
| *Efficiency* | *7* | *50.0* | *4* | *28.6* | *2* | *14.3* | | | | | *1* | *7.1* |
| *Security/Integrity* | *7* | *50.0* | *3* | *21.4* | *3* | *21.4* | *1* | *7.1* | | | | |

*Notable Keys: EX = Excellent; V.G = Very Good; % of = Percentage of*

**Table 9. Also the above table reflects the data gathered from the respondents who are interested in optimizing one or more software qualities in their software systems.**

| Indication of interest in optimizing siftware qualities | | | | |
|---|---|---|---|---|
| **Qualities (S/N)** | **Yes, I want it optimized** | **Percentage of yes** | **No, it is not necessary** | **Percentage of no** |
| Understandability | 9 | 64.3 | 5 | 35.7 |
| Completeness | 7 | 50.0 | 7 | 50.0 |
| Conciseness | 7 | 50.0 | 7 | 50.0 |
| Interoperability | 9 | 64.3 | 5 | 35.7 |
| Reusability | 6 | 42.9 | 8 | 57.1 |
| Integrability | 8 | 57.1 | 6 | 42.9 |
| Portability | 8 | 57.1 | 6 | 42.9 |
| Consistency | 9 | 64.3 | 5 | 35.7 |
| Maintainability | 10 | 71.4 | 4 | 28.6 |
| Testability | 7 | 50.0 | 7 | 50.0 |
| Usability | 9 | 64.3 | 5 | 35.7 |
| Reliability | 9 | 64.3 | 5 | 35.7 |
| Efficiency | 9 | 64.3 | 5 | 35.7 |
| Security/Integrity | 10 | 71.4 | 4 | 28.6 |

The following table indicates the number of power user companies who prefer a better software package(s)

**Table 10. Signifying users who prefer a better software packages**

| | **Frequency** | **Percentage** |
|---|---|---|
| Yes | 8 | 57.1 |
| No | 6 | 42.9 |
| Total | 14 | 100.0 |

The following table indicates the number of power users companies who would want to change their software developers due to one reason or the other;

**Table 11. Indicating the power user company having interest in changing their software developers**

| | **Frequency** | **Percentage** |
|---|---|---|
| Yes | 1 | 7.1 |
| No | 13 | 92.9 |
| Total | 14 | 100.0 |

At the end of the informal interview, I was able to confirm that the power users' point of view of the software systems is close to what their vendors (software developers) said about the performance and functionalities of software systems. The software developers have always been found to be exaggerative about their products and would always want to preach good messages about their products. Hence, the power users' inherently share their view about software performance because both parties always work collaboratively to maintain their software products.

On the other hand, common user's views are conflicting to the software developers and their corresponding power users'. This is because they are always at the receiving end of any lapses, mistakes, error and misconception undertaken during every software development project. As such, they are very excited to relay information about their software failures to any researcher who wants to embark on a project relating to software failures and optimization.

As a result of the above reasons, I concluded in using the common users' view and analysis as the basis for the research. I later proceeded to identifying the supportive factors and problems associated with each software quality. Using the requirement negotiation processes embedded in force field analysis, I was able to establish a bench-mark that forms a foundational model for every software development project.

### 10.6 Validity of Research Instruments

The interviews questions and questionnaires were presented to the researcher's supervisor that made certain modifications, corrections and directives, which were effected before the research instrument was finally administered.

### 10.7 Result and Interpretation

In the section 10.1, I explained different views of Software Stakeholders: The Software Developers, Software Power Users and the Common Users. I also gave the definition of Power Users as the category of users who are making use of the software systems to render services for the populates and the common users as those who are making use of the services rendered by the power users.

Furthermore, I narrated my experiences as regards the opinions of different stakeholders. The power users often shared "inherited views and beliefs" about software systems with developers because both of them also shared the responsibility of maintaining software systems. As such, power users are in most cases believe whatever their developers say about the software systems.

On the contrary, the common users had conflicting beliefs about the performance of the software systems. This was shown from the analysis of the data which revealed that most common users are not satisfied with the current level of performance of the existing software systems.

I therefore deduced from the findings that the common users had honest views on the performance of software systems because they are always at the receiving end. They notice every lapses, failure, weaknesses, and mistakes incurred in the software systems. As a result of the above reasons, I decided to walk in line with the views of the common users.

My proposed model of software development embraces the use of win-win software development model. This is because it is only the win-win software development model that incorporates the requirement negotiation process which is a significant tool in bridging the gap between the user's views, I decided to map these views into the second sector of win-win spiral model.

I believe the most effective way of solving the enumerated problem of software systems' performance is to identifying the users motivations, needs and desires as well as the software developers' priority, using the requirement negotiation process embedded in win-win software development model, it would be easy to shed off the unprofessional practices often indulged by the power users as well as unethical pattern of usage of the common users. Also, using this model one can reinforce the software development practices and pattern of usage that support the effective and efficient performance of software systems. Before discussing on the use of win-win software development model, I will like to give some explanations on this model.

## 11 Details of Win-win Spiral Software Development Model

The developed Win-win Spiral Model [8] uses the theory W (win-win) approach [6] to converge on a system's next-level objectives, constraints, and alternatives. In Fig. 1, The Theory W approach involves identifying the system's stakeholders and their win conditions, and using requirement negotiation process to determine a mutually satisfactory set of objectives, constraints, and alternatives for the stakeholders. The following step reflects the mode of operation of win-win spiral model of software development.

## 11.1 The Win-win Spiral Model of Software Development

### 11.1.1 Determine objectives

Identify the system life-cycle stakeholders and their win conditions. Establish initial system boundaries and external interfaces.

### 11.1.2 Determine constraints

Determine the conditions under which the system would produce win-lose or lose-lose outcomes for some stakeholders.

### 11.1.3 Identify and evaluate alternatives

Solicit suggestions from stakeholders. Evaluate them with respect to stakeholders' win conditions. Synthesize and negotiate candidate win-win alternatives. Analyze, Asses, and resolve win-lose or lose-lose risks. Record Commitments and areas to be left flexible, in the project's design record and life cycle plans.

### 11.1.4 Cycle through the spiral

Elaborate win conditions, screen alternatives, resolve risks, accumulate appropriate commitments, and develop and execute downstream plans.

Fig. 1 Illustrates the win-win Spiral Model. The original Spiral Model had four sectors, beginning with "Establish next-level objectives, constraints, alternatives." The two additional sectors in each spiral cycle, i.e "Identifying next-level Stakeholders" and "Identifying Stakeholders' win Conditions", and the "Reconcile win Conditions" portion of the third sector, provide the collaborative foundation for the model. They also fill a missing portion of the original Spiral Model: the means to answer, "Where do the next-level objectives and constraints come from, and how do you know they are the right ones?" The refined Spiral Model also explicitly addresses the need for concurrent analysis, risk resolution, definition, and elaboration of both the software product and the software process.

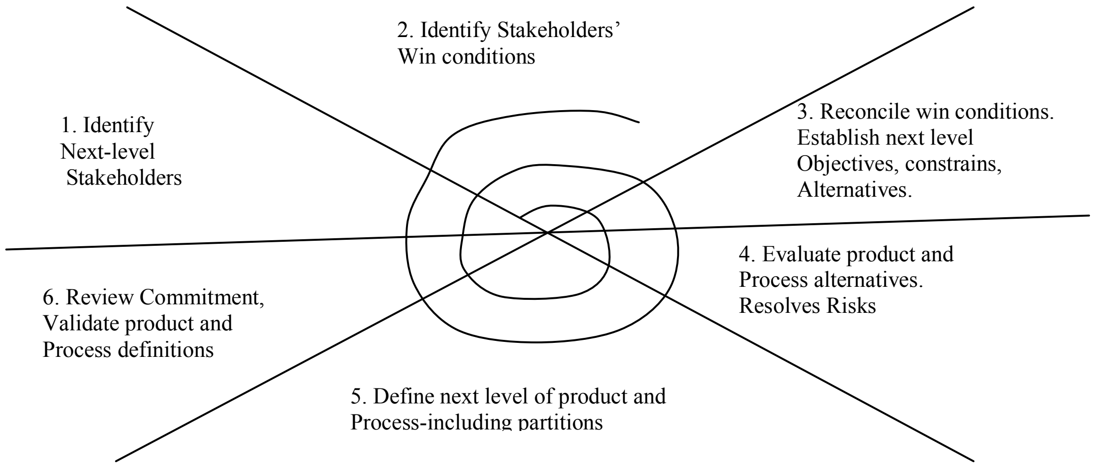

2. Identify Stakeholders' Win conditions
1. Identify Next-level Stakeholders
3. Reconcile win conditions. Establish next level Objectives, constrains, Alternatives.
4. Evaluate product and Process alternatives. Resolves Risks
6. Review Commitment, Validate product and Process definitions
5. Define next level of product and Process-including partitions

**Fig. 1. The win-win spiral model**

### 11.1.5 My main work in this research starts from the second to fourth sector of the spiral

I have been able to identify the stakeholders' win conditions in my field survey (which is the second sector of the spiral). In order to reconcile the stakeholders' win conditions, establish the next level objectives, constraints and alternatives, I introduced the concept of Force Field Analysis which views the requirement negotiation process as a system comprising two main forces (Driving Forces and Restraining Forces). This process constitutes the impact of this research on the third sector of win-win spiral model of software development.

### 11.2 Resolving win Conditions- Establish Next Level Objectives, Constraints and Alternatives

As specified before, the third and the fourth sector would be looked into in this section. Here, we tried to identify the supporting factors and the constraints associated with each software quality highlighted and used requirement negotiation process to establish a benchmark (win-win condition) that satisfies the developers and the users via force field analysis. Before discussing how it was used in this project, I will like to explain the concept of force field analysis.

## 12 Incorporating Force Field Analysis and Its Concepts

Force Field Analysis was a concept firstly pioneered by a Social Psychologist, Lewin Kurt [5]. It was later incorporated in Physics to show-case the supporting forces and restraining forces against a proposed project. It is an important concept mostly used by project managers for building an understanding of the forces that will drive or resist a proposed change. Force Field Analysis helps to weigh the importance of the factors relating to a particular decision and to know whether a plan is worth implementing. It is a specialization method of weighing the pro and cons.

By carrying out the analysis, one can plan to strengthen the forces supporting a decision, and reduce the impact of opposition to it.

### 12.1 A Conceptual Semantics of Force Field Analysis

The analysis of a force field can be made easy using a force field flow chat. The chat is derived from the work of social psychologist, Lewin Kurt. Acoording to Lewin's theories, human behavior is caused by forces, beliefs, expectations, cultural norms, and the like-within the "life space" of an individual or society. These forces can be positive, urging us towards a behavior, or negative, propelling us away from a behavior. A force field diagram portrays these driving forces and restraining forces that affect a central question or problem.

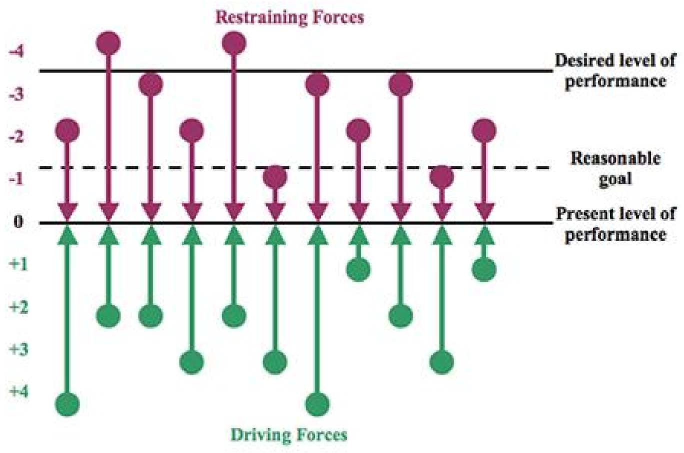


**Fig. 2. Lewin kurt's model on force-field analysis**

### 12.1.1 Driving forces

Driving forces are those forces affecting a situation that are pushing a particular decision; they tend to initiate a change and keep it going. Improving productivity in a work group, pressure from a supervisor, incentives earnings, and competition may be examples of driving forces.

### 12.1.2 Restraining forces

Restraining forces are forces acting to restrain or decrease the driving forces. Apathy, hostility and poor maintenance of equipment may be example of restraining forces against increased production.

### 12.1.3 Equilibrium

Equilibrium is reached when a sum of the driving forces equal is the sum of the restraining forces. In my example, equilibrium represents the present level of productivity, as shown in Fig. 3 below;

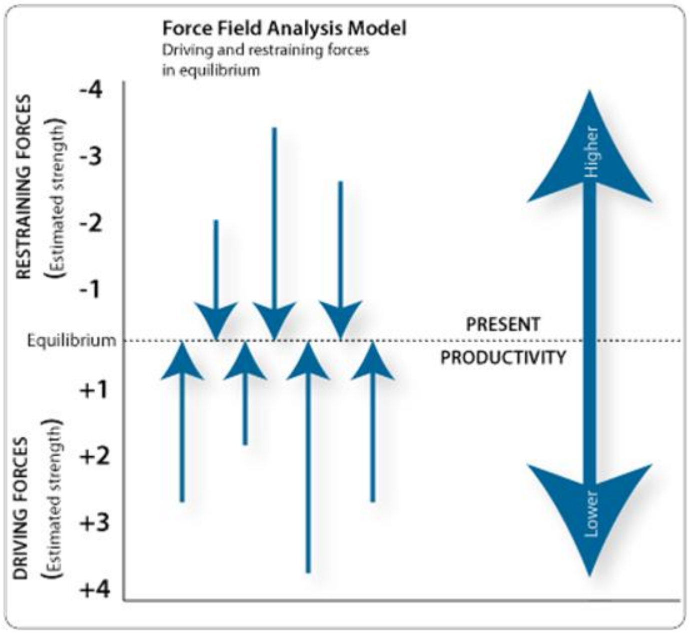
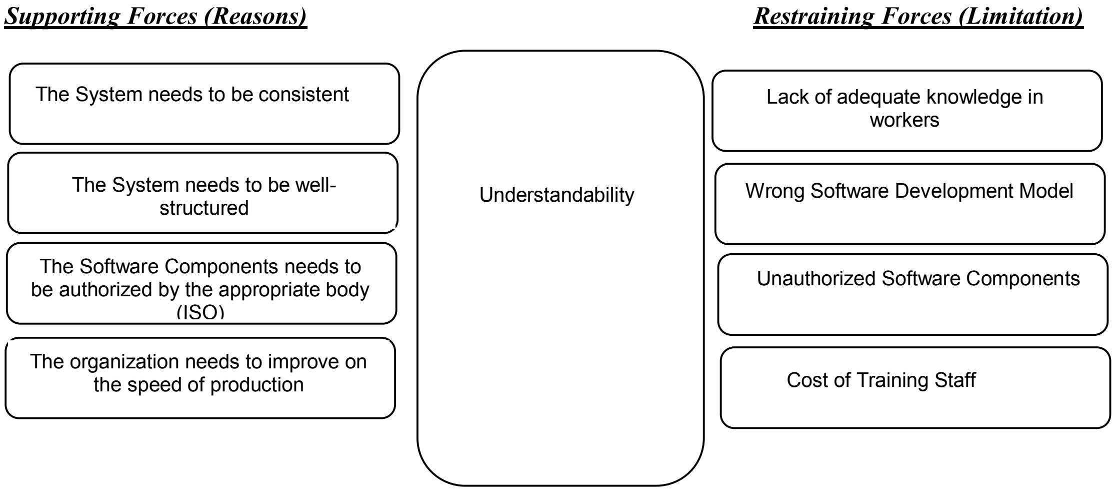


**Fig. 3. A scientific interpretation of Lewin Kurt's model**

This equilibrium, or present level of productivity, can be raised or lowered by changes in the relationship between the driving and restraining forces. In order to relate the concept of force field analysis to the on-going project, my procedure demands looking into those software features and qualities in today's software systems one-by-one and apply force field analysis concept aimed at improving the necessary software qualities in the examined software package or system. The force field analysis is to determine the supporting forces and restraining forces that are significantly affecting the development of a good quality software system. Force field analysis can help to improve the quality of a software system in two ways:

i. Reduce the strength of the forces opposing a software project
ii. Increase the strength of the supportive forces of a software project

I proceeded by examining the necessary software qualities together with their associated win-conditions one by one and apply force field analysis concept on them. On each side of the force field analysis, I attach a total of eight (8) points (*e.g eight-point value is attached to the supporting force(s) and the restraining* *force(s)of the force field analysis respectively*). A mathematical notation was later developed to display the overall negotiation process on the restraining forces and the overall improvement sequel to the adoption of force field analysis. I then concluded the research by giving a framework or model of how each of the examined software systems should be designed and developed, suggesting the procedures to follow before, during and after the software development process. The first software quality to examine is **Understandability.** Force field Analysis can be applied to this quality as follows;


Supporting Forces (Reasons)
Restraining Forces (Limitation)
The System needs to be consistent
The System needs to be well-structured
The Software Components needs to be authorized by the appropriate body (ISO)
The organization needs to improve on the speed of production
Understandability
Lack of adequate knowledge in workers
Wrong Software Development Model
Unauthorized Software Components
Cost of Training Staff

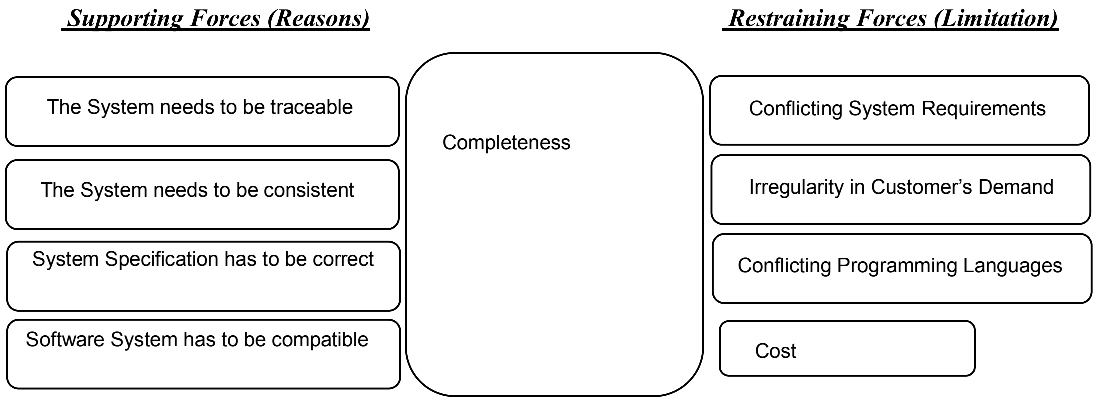
If we are to assign eight points to both the supporting forces and the restraining forces respectively. The analysis goes as follows:

- Adequate training for staff increases cost but also enhances knowledge of staff **(+1,-1, strength to the restraining forces but also take care of the limitation of unskilled workers)**
- Improve speed of production and reduce work overtime **(+1, strengthen to the supporting forces)**
- Use of software components authorized by the appropriate organization (ISO) only by the software developers and experts that are skilled enough in developing a software package will take care of unauthorized software components and the engagement of unskilled workers**, (-2, reduce the strength of the restraining forces)**

***The Result:***

In the end, the ratio between the supporting forces and restraining forces on software understandability has changed through this analysis from 8:8 to 10:7. **Hence, there is a considerable improvement**.

Second on the list is Completeness


Supporting Forces (Reasons)
Restraining Forces (Limitation)
The System needs to be traceable
The System needs to be consistent
System Specification has to be correct
Software System has to be compatible
Completeness
Conflicting System Requirements
Irregularity in Customer's Demand
Conflicting Programming Languages
Cost

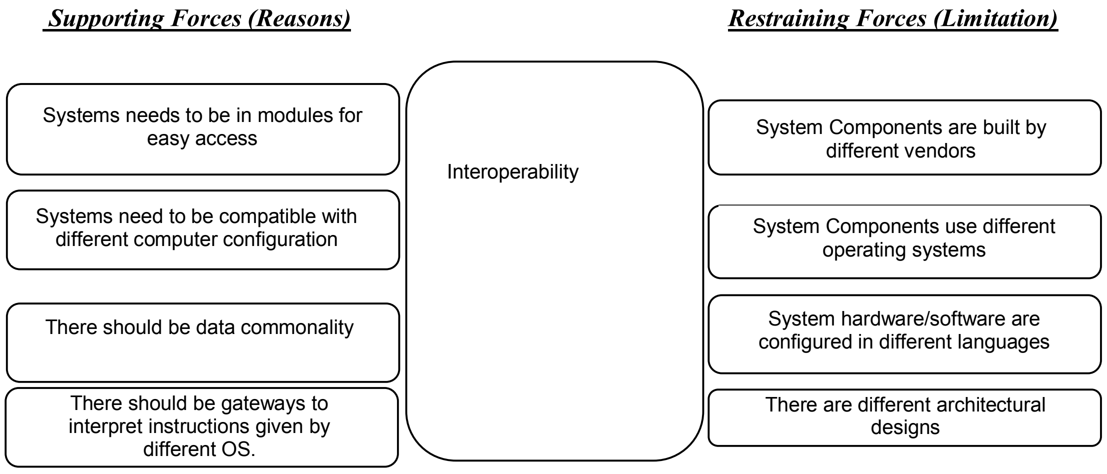
If we are to assign eight points to both the supporting forces and the restraining forces respectively. The analysis goes as follows:

- ❖ It is costly to make a complete software product **(+1, reinforcing the restraining forces)**
- ❖ Making use of win-win software development model with the embedded software requirement negotiation process in the model will checkmate irregularities in customer's demands **(-1, reducing the strength of the restraining forces)**
- ❖ Making use of software component authorized by international standard organization (ISO) will take care of different vendor specification because they are all authorized by the same authorizing body and as such, will accommodate or permit the same range of functional requirements **(+1,-1 reducing the strength of the restraining forces and increasing the strength of the supporting forces)**
- ❖ Making use of a software development model that will accommodate a comprehensive deliverable for each stage of software development will help refine the needs and the requirements (i.e *specification*) accurately and will also enhance consistency, **(+2, reinforcing the supporting forces)**
- ❖ Making use of a compatible programming languages or conventional programming languages that has extensive library facilities will help incorporate old programming languages **e.g Fortran 66-to-77-to Fortran 7x et.c(-1, reducing the restraining forces)**

***The Result***

In the end, the ratio between the supporting forces and restraining forces on software completeness has changed through this analysis from 8 :8 to 12 : 5 in favor of the supporting forces. ***Hence, there is a considerable improvement***

Third on the list is **interoperability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Systems needs to be in modules for easy access
Systems need to be compatible with different computer configuration
There should be data commonality
There should be gateways to interpret instructions given by different OS.
Interoperability
System Components are built by different vendors
System Components use different operating systems
System hardware/software are configured in different languages
There are different architectural designs

*The Analysis*

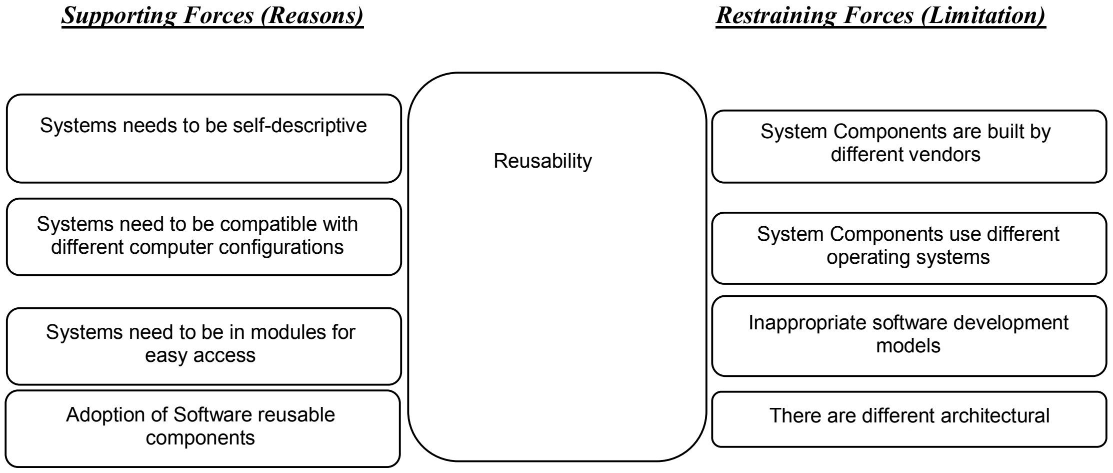
If we are to assign eight points to both the supporting forces and the restraining forces respectively. The analysis goes as follows:

- ❖ Making use of a software packages using the same or upgrade operating systems e.g Window 2003-to-2007-to-Vista, **(-1, reducing the strength of the restraining forces)**
- ❖ Installation of Gateway software for the conversion and translation of various programming languages and operating systems into comprehensive and understandable ones **(+1,-1, reducing the strength of the restraining forces and increasing the strength of the supporting)**

- ❖ Making use of a compatible programming languages or conventional programming languages that has extensive library facilities will help incorporate old programming languages e.g Fotran 66-to-Fortran 77-to-Fortran 7x et c., **(-1, reducing the strength of the restraining forces)**
- ❖ Making use of software component authorized by international standard organization (ISO) will take care of different vendor specification because they are all authorized by the same authorizing body and as such, will accommodate or permit the same range of functional requirements **(-1,+1, reducing the strength of the restraining forces and increasing the strength of the supporting forces)**

***The Result:*** In the end, the ratio between the supporting forces and restraining forces on software interoperability has changed through the analysis from 8:8 to 10 : 4 in favor of the supporting forces. *Hence, there is a considerable improvement*

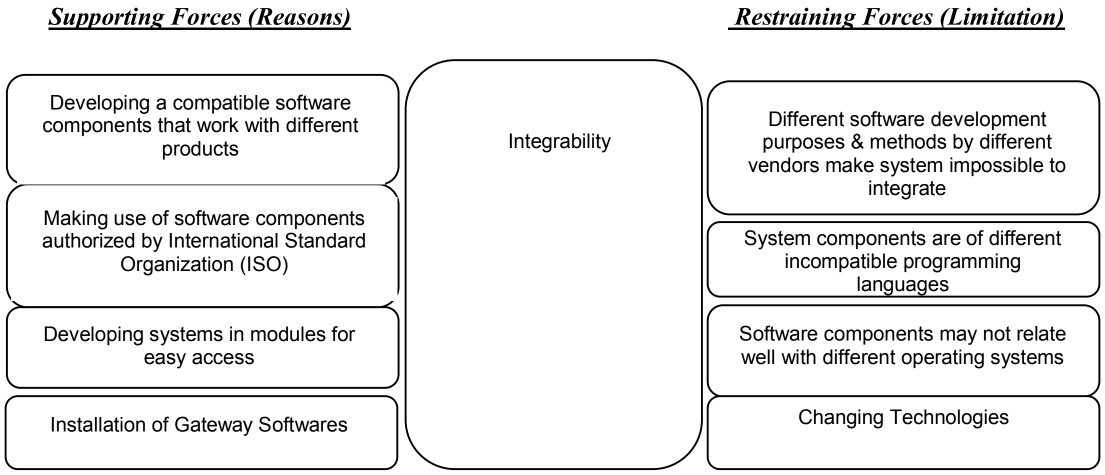
Fourth on the list is **Reusability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Systems needs to be self-descriptive
Systems need to be compatible with different computer configurations
Systems need to be in modules for easy access
Adoption of Software reusable components
Reusability
System Components are built by different vendors
System Components use different operating systems
Inappropriate software development models
There are different architectural

*The Analysis*

- ➢ Making use of intelligent software agents **(+2, reinforcing the supporting forces)**
- ➢ Making use of software components authorized by international standard organization (ISO) will take care of different vendor specification because they are all authorized by the same authorizing body and as such, will accommodate or permit the same range of functional requirements **(-1,+1, reducing the strength of the restraining forces and increasing the strength of the supporting forces)**
- ➢ Making use of a software packages using the same or upgradable operating systems e.g. Windows 2003-to-2007-to-Vista **(-1, reducing the strength of the restraining forces)**
- ➢ Installation of Gateway software for the conversion and translation of various programming languages and operating systems into comprehensive and understandable ones **(-1,+1, reducing the strength of the restraining forces and increasing the strength of the supporting forces)**
- ➢ Making use of a win-win software development model with appropriate delivery for each stage of the developmental process will make the system accessible in modules **(+1, reinforcing the supporting forces)**

Fifth on the list is **Integrability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Developing a compatible software components that work with different products
Making use of software components authorized by International Standard Organization (ISO)
Developing systems in modules for easy access
Installation of Gateway Softwares
Integrability
Different software development purposes & methods by different vendors make system impossible to integrate
System components are of different incompatible programming languages
Software components may not relate well with different operating systems
Changing Technologies

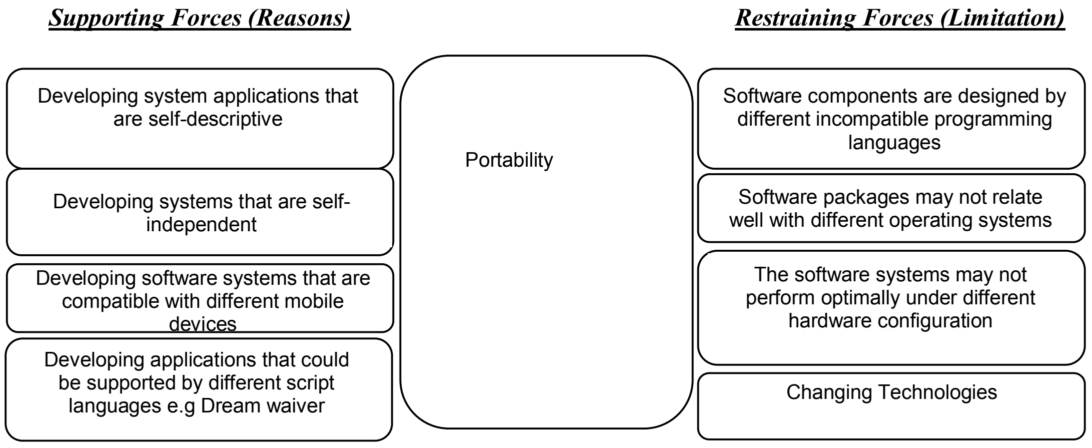
*The Analysis*

If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- Making use of software components authorized by International Standard Organization (ISO) will take care of different vendor specification because they are all authorized by the same authorizing body and as such, will accommodate or permit the same range of functional requirements **(-1, reducing the strength of restraining forces)**
- Installation of Gateway Software **(-1,+1, reducing the strength of the restraining forces and increasing the strength of the supporting forces)**
- **M**aking use of an updated software development method and CASE tools e.g Web-centric software development method **(-1, reducing the strength of the restraining forces)**

The Result
In the end, the ratio between the supporting forces and the restraining forces on software integrability has changed through this analysis ratio 8:8 to 10:5 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

Sixth on the list is **Portability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Developing system applications that are self-descriptive
Developing systems that are self-independent
Developing software systems that are compatible with different mobile devices
Developing applications that could be supported by different script languages e.g Dream waiver
Portability
Software components are designed by different incompatible programming languages
Software packages may not relate well with different operating systems
The software systems may not perform optimally under different hardware configuration
Changing Technologies

*The Analysis*

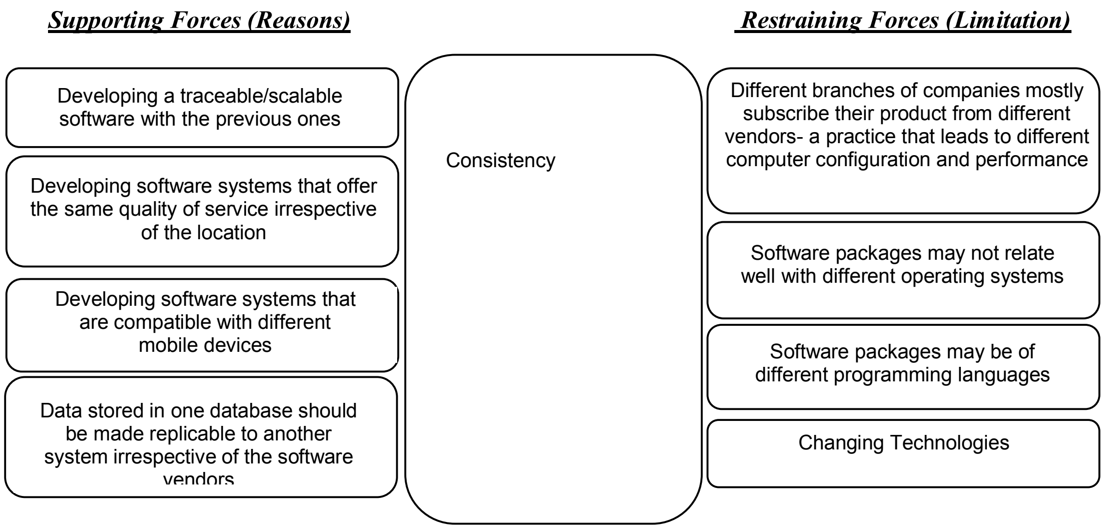
If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- ➢ Making use of intelligent software agents will enhance certain level of autonomy in software systems **(+1, reinforcing the supporting forces)**
- ➢ Making use of software component authorized by international standard organization (ISO) will take care of different vendor specification because they are all authorized by the same authorizing body and as such, will accommodate or permit the same range of functional requirements **(+1, increasing the strength of the supporting forces)**
- ➢ Installation of Gateway software **(-1, reducing the strength of the limiting forces)**
- ➢ Installation of the appropriate software user-interface **(-1, reducing the strength of the limiting forces)**

**The Result**

In the end, the ratio between the supporting forces and the restraining forces on software Portability has changed through this analysis ratio 8:8 to 10:6 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

The seventh quality is **Consistency**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Developing a traceable/scalable software with the previous ones
Developing software systems that offer the same quality of service irrespective of the location
Developing software systems that are compatible with different mobile devices
Data stored in one database should be made replicable to another system irrespective of the software vendors
Consistency
Different branches of companies mostly subscribe their product from different vendors- a practice that leads to different computer configuration and performance
Software packages may not relate well with different operating systems
Software packages may be of different programming languages
Changing Technologies

*The Analysis*

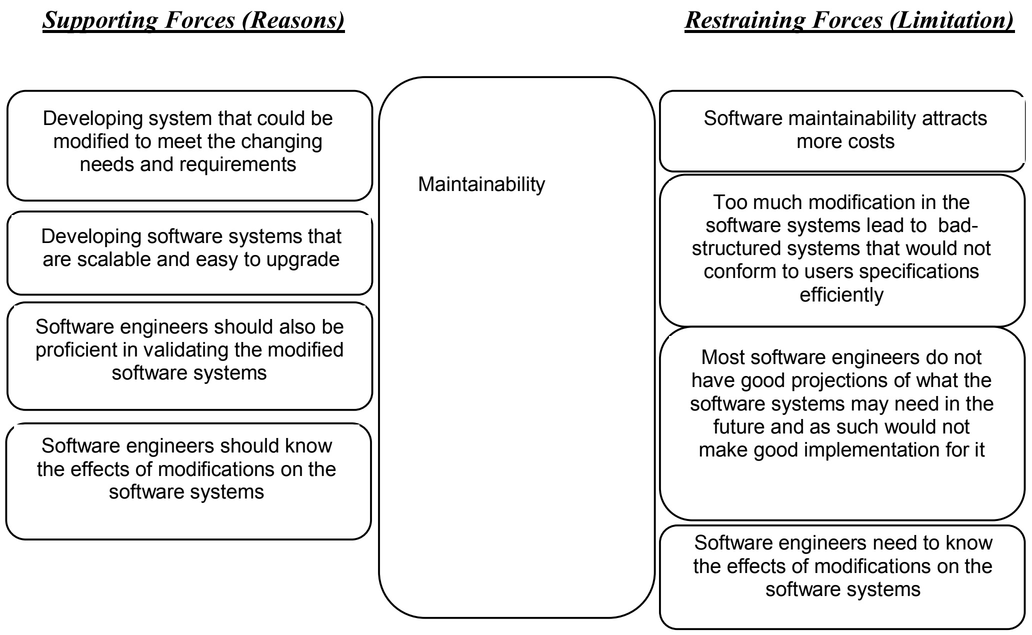
If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- ➢ Making use of a conventional programming languages that could be easily be upgraded e.g. **C**-to - **C+** - to - **C# (-1, reducing the strength of the limiting forces)**
- ➢ Installation of gateway software and appropriate compilers **(-1, reducing the strength of the limiting forces)**
- ➢ Making use of software components authorized by international standard organization (ISO) will take care of different vendor specification because they are all authorized by the same authorized by the same authorizing body and as such, will accommodate or permit the same range of functional requirements**(-1,+1, reducing the strength of the restraining forces and increasing the strength of the supporting forces)**

**The Result**

In the end, the ratio between the supporting forces and the restraining forces on software Consistency has changed through this analysis ratio 8:8 to 9:5 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

The Eighth Quality is **Maintainability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Developing system that could be modified to meet the changing needs and requirements
Developing software systems that are scalable and easy to upgrade
Software engineers should also be proficient in validating the modified software systems
Software engineers should know the effects of modifications on the software systems
Maintainability
Software maintainability attracts more costs
Too much modification in the software systems lead to bad-structured systems that would not conform to users specifications efficiently
Most software engineers do not have good projections of what the software systems may need in the future and as such would not make good implementation for it
Software engineers need to know the effects of modifications on the software systems

*The Analysis*

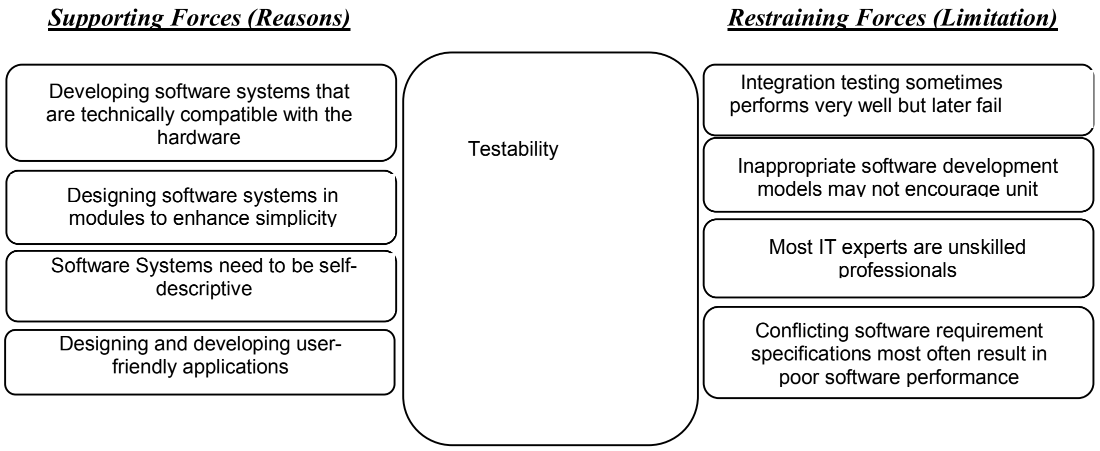
If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- More training for the development staff on software modification. This attracts more cost and bring extra expenses **(-1, strengthening the restraining forces, but also enhancing the supporting forces, +1)**
- Making use of the appropriate CASE tools **(Reinforcing the supporting forces, +1)**
- Making use of updated web-centric software development method would enable easy software refactoring **(This also reinforces the supporting factor, +1)**
- Making use of a software product or system built in the same computer configuration **(Alleviating the restraining forces, -1)**

***The Result***

In the end, the ratio between the supporting forces and the restraining forces on software Maintainability has changed through this analysis ratio 8:8 to 11:6 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

The Ninth Quality is **Testability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Developing software systems that are technically compatible with the hardware
Designing software systems in modules to enhance simplicity
Software Systems need to be self-descriptive
Designing and developing user-friendly applications
Testability
Integration testing sometimes performs very well but later fail
Inappropriate software development models may not encourage unit
Most IT experts are unskilled professionals
Conflicting software requirement specifications most often result in poor software performance

*The Analysis*

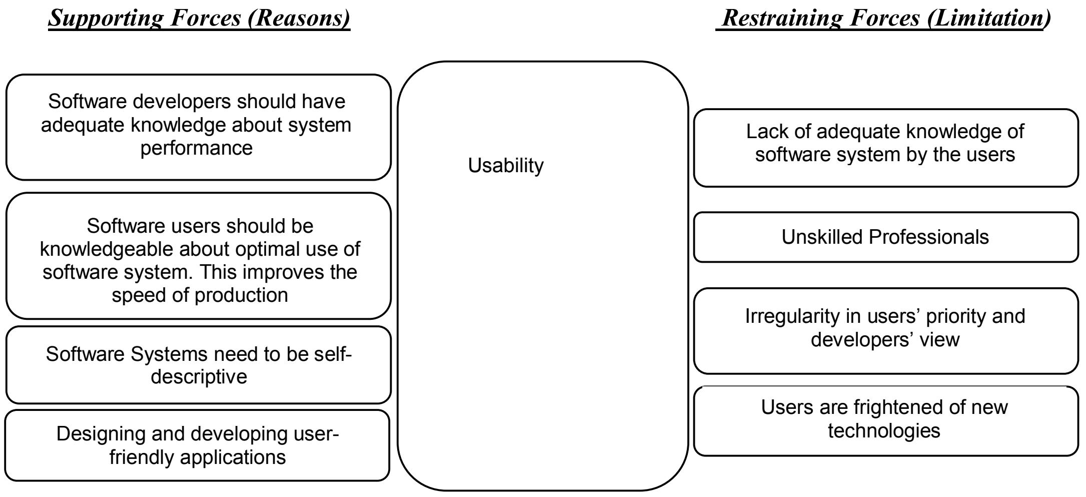
If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- Adopting win-win software development model which demands requirement negotiation process will resolve conflicting software requirements **(-1, reducing the strength of the limiting forces)**
- The software industries should encourage a development model that requires unit testing for various software components before integration testing **(-1, reducing the strength of the limiting forces)**
- Making use of intelligent software agents **(+1, reinforcing the supporting forces)**
- Adopting well-structured programming languages to enhance modularity **(+1, reinforcing the supporting forces)**

***The Result***

In the end, the ratio between the supporting forces and the restraining forces on software Testability has changed through this analysis ratio 8:8 to 10:6 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

The Tenth quality is **Usability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Software developers should have adequate knowledge about system performance
Software users should be knowledgeable about optimal use of software system. This improves the speed of production
Software Systems need to be self-descriptive
Designing and developing user-friendly applications
Usability
Lack of adequate knowledge of software system by the users
Unskilled Professionals
Irregularity in users' priority and developers' view
Users are frightened of new technologies

*The Analysis*

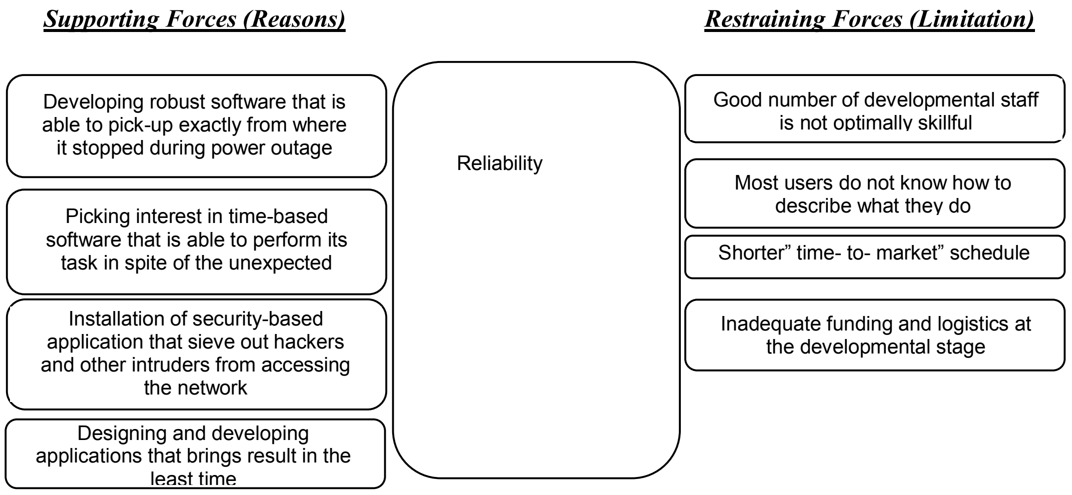
If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- Adequate training will enhance knowledge and improve skills in developmental staff **(-2, reducing the strength of the restraining forces and increasing the strength of the supporting forces, +1)**
- Making use of recent CASE tools and software development method that allows users and clients to view the software development growth (e.g Web- centric software development) and hence contribute **(-1,+1, reducing the strength of the restraining forces and increasing the strength of the supporting forces)**
- Application of self-descriptive, sensitive and intelligent software agents **(+1, reinforcing the supporting forces)**

The Result

In the end, the ratio between the supporting forces and the restraining forces on software Usability has changed through this analysis ratio 8:8 to 11:5 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

The Eleventh quality is **Reliability**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Developing robust software that is able to pick-up exactly from where it stopped during power outage
Picking interest in time-based software that is able to perform its task in spite of the unexpected
Installation of security-based application that sieve out hackers and other intruders from accessing the network
Designing and developing applications that brings result in the least time
Reliability
Good number of developmental staff is not optimally skillful
Most users do not know how to describe what they do
Shorter" time- to- market" schedule
Inadequate funding and logistics at the developmental stage

*The Analysis*

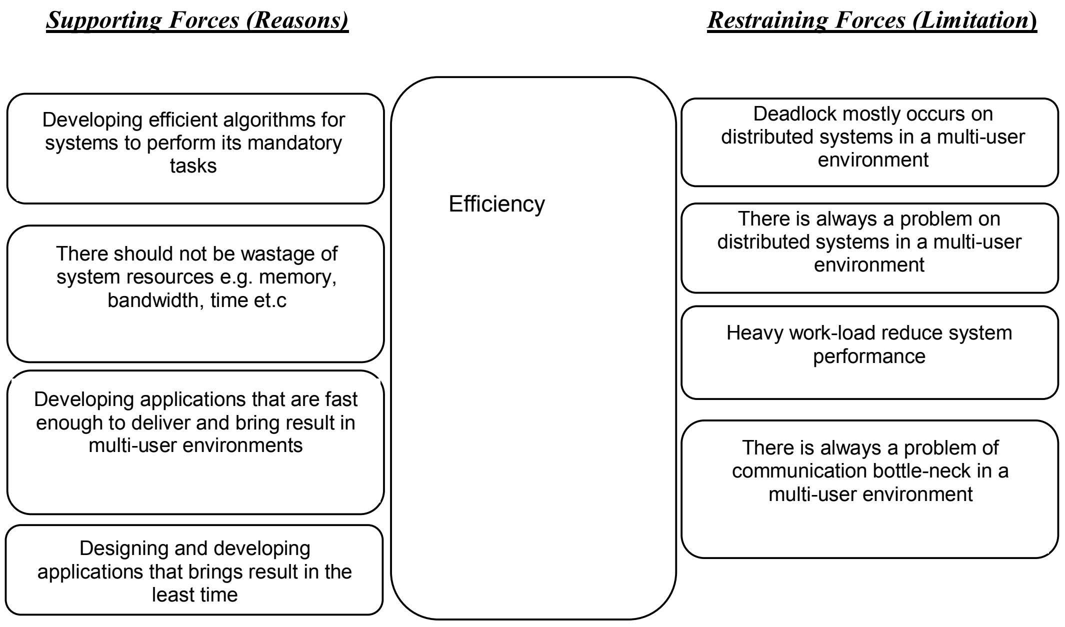
If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- Adequate training for software developers and prospective users **(+1,-1, raising the strength of the supporting forces but also reducing the strength of the restraining forces)**
- Making use of deliverables that will give good account of each stage of developmental process will enhance accuracy in the software performance **(+1, reinforcing the supporting forces)**
- Engage in a win-win software developmental process that facilitate software requirement negotiation activities **(-1, reducing the strength of the limiting forces)**
- Incorporation and installation of security-based applications that protect hackers and other intruders would help sieve out instability in software performance **(+1, reinforcing the supporting forces)**

***The Result:***

In the end, the ratio between the supporting forces and the restraining forces on software Reliability has changed through this analysis ratio 8:8 to 11:6 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

The twelfth quality is **Efficiency**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Developing efficient algorithms for systems to perform its mandatory tasks
There should not be wastage of system resources e.g. memory, bandwidth, time et.c
Developing applications that are fast enough to deliver and bring result in multi-user environments
Designing and developing applications that brings result in the least time
Efficiency
Deadlock mostly occurs on distributed systems in a multi-user environment
There is always a problem on distributed systems in a multi-user environment
Heavy work-load reduce system performance
There is always a problem of communication bottle-neck in a multi-user environment

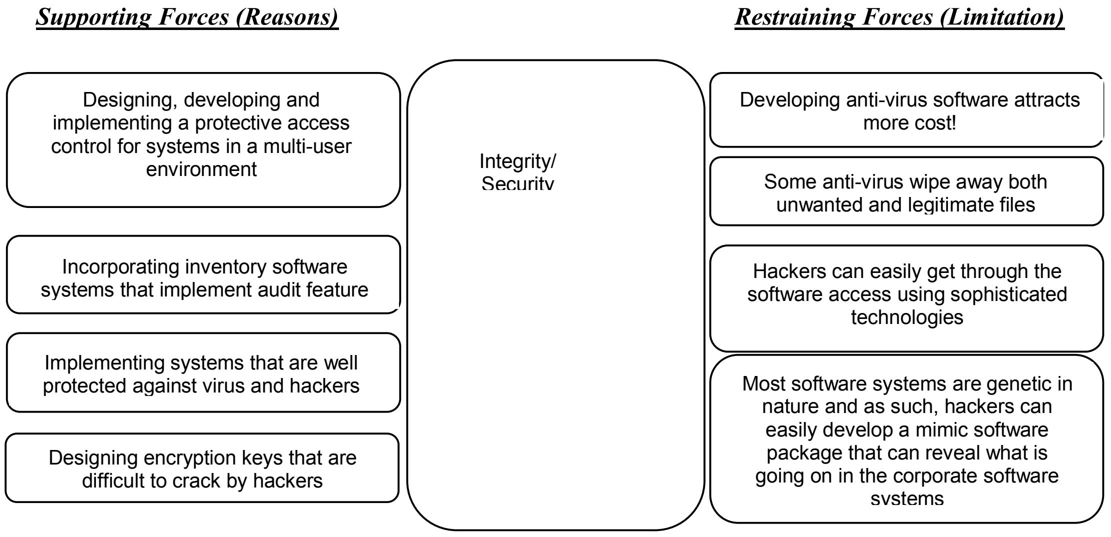
If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- Designing and implementing a wait-for-graph to detect and break the possible "deadlocks" **(-1, reducing the strength of the limiting forces)**
- Making use of a 3-phase commit protocol **(-1, reducing the strength of the limiting forces)**
- De-centralization of database to enhance a certain level of autonomy within each distributed system **(-1, reducing the strength of the limiting forces)**
- The design and development of appropriate compilers for looping optimization that would enable software programs run faster **(-1, reducing the strength of the limiting forces)**

***The Result***

In the end, the ratio between the supporting forces and the restraining forces on software Efficiency has changed through this analysis ratio 8:8 to 8:4 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

The Thirteenth quality is **Integrity/Security**


Supporting Forces (Reasons)
Restraining Forces (Limitation)
Designing, developing and implementing a protective access control for systems in a multi-user environment
Incorporating inventory software systems that implement audit feature
Implementing systems that are well protected against virus and hackers
Designing encryption keys that are difficult to crack by hackers
Integrity/ Security
Developing anti-virus software attracts more cost!
Some anti-virus wipe away both unwanted and legitimate files
Hackers can easily get through the software access using sophisticated technologies
Most software systems are genetic in nature and as such, hackers can easily develop a mimic software package that can reveal what is going on in the corporate software systems

*The Analysis*

If we are to assign eight points to both supporting forces and restraining forces respectively. The analysis goes as follows:

- Developing anti-virus to protect hackers' attack **(-1,+1, reducing the strength of the restraining forces and increasing the strength of the supporting forces)**
- Making use of a wireless technology like WLAN with good security features reduce unnecessary interruption from hackers **(+1, reinforcing the supporting forces)**
- Software users should be assigned a username and a password only to legitimate users and recognized by the software systems **(-1, reducing the strength of the limiting forces)**
- Making use of a "Taylor-made" (customized) software components during the development stage would protect the software system from being easily predicted by outsiders who may want to know what is going on within the organization **(-1, reducing the strength of the limiting forces)**
- Developing a strong fire-wall would prevent unnecessary entrance into a sensitive part of the software systems **(-1, reducing the strength of the limiting forces)**

**The Result**

In the end, the ratio between the supporting forces and the restraining forces on software Efficiency has changed through this analysis ratio 8:8 to 10:4 in favor of the supporting forces. ***Hence, there is a considerable improvement.***

## 13 Discussion of Result(s)

For each software quality, the abbreviated terms are represented as follows; Software quality for understandability before it was subjected to force field analysis is represented as; $Q_{U1(i)}$, after the analysis $Q_{U2(f)}$, and the resultant improvement is represented as $Q_{U1(I)}$.

It therefore goes for other qualities as shown in the following table;

| S/N | Software quality | Before the analysis | After the analysis | Resultant improvement |
|---|---|---|---|---|
| 1 | Understandability | $QU_{1(i)}$ | $QU_{1(f)}$ | $QU_1(I)$ |
| 2 | Completeness | $QC_{1(i)}$ | $QC_{1(f)}$ | $QC_1(I)$ |
| 3 | Interoperability | $QI_{1(i)}$ | $QI_{1(f)}$ | $QI_1(I)$ |
| 4 | Reusability | $QR_{(i)}$ | $QR_{(f)}$ | $QR(I)$ |
| 5 | Integrability | $QI_{2(i)}$ | $QI_{2(f)}$ | $QI_2(I)$ |
| 6 | Portability | $QP_{(i)}$ | $QP_{(f)}$ | $QP(I)$ |
| 7 | Consistency | $QC_{2(i)}$ | $QC_{2(f)}$ | $QC_2(I)$ |
| 8 | Maintainability | $QM_{(i)}$ | $QM_{(f)}$ | $QM(I)$ |
| 9 | Testability | $QT_{(i)}$ | $QT_{(f)}$ | $QT(I)$ |
| 10 | Usability | $QU_{2(i)}$ | $QU_{2(f)}$ | $QU_2(I)$ |
| *11* | *Reliability* | $QR_{(i)}$ | $QR_{(f)}$ | $QR(I)$ |
| 12 | Efficiency | $QE_{(i)}$ | $QE_{(f)}$ | $QE(I)$ |
| 13 | Integrity/Security | $QIS_{(i)}$ | $QIS_{(f)}$ | $QIS(I)$ |

In line with the force field analysis model developed, it follows that the default value attached to Understandability is 8:8 (*for both the supporting and the restraining force(s)*),

- Hence, $QU_{1(i)}$ = 8:8; and $QU_{1(f)}$ = 10:7
  Therefore, $QU_1(I)$ is however ($QU_{1(f)}$ - $QU_{1(i)}$ ) = (10/7 – 8/8) = 4/14
- Again, for the second software quality Completeness $QC_{1(i)}$ = 8:8 and $QC_{1(f)}$ = 12:5
  $QC_1(I)$ = $QC_{1(f)}$ - $QC_{1(i)}$ = (12/5 – 8/8) = 7/5
- For the third software quality, Interoperability, $QI_{1(i)}$ = 8:8; $QI_{1(f)}$ = 10:4
  $QI_1(I)$ = ($QI_{1(f)}$ - $QI_{1(i)}$) = (5/2 -1) = 3/2
- For the fourth software quality, Reusability, $QR_{(i)}$ = 8:8; $QR_{(f)}$ = 13:5
  $QR(I)$ = ($QR_{(f)}$ - $QR_{(i)}$) = (13/5 – 8/8) = 9/5
- The fifth software quality is Integrability; $QI_{2(i)}$ = 8:8 ; $QI_{2(f)}$ = 10:5
  $QI_2(I)$ = ($QI_{2(f)}$ - $QI_{2(i)}$) = (2-1) = 1
- For the sixth software quality; Portability, $QP_{(i)}$ = 8:8, $QP_{(f)}$ = 10:6
  $QP(I)$ = ($QP_{(f)}$ - $QP_{(i)}$) = (5/3 – 1) = 2/3
- The seventh software quality is Consistency; and $QC_{2(i)}$ = 8:8, $QC_{2(f)}$ = 9:5
  $QC_2(I)$ = ($QC_{2(f)}$ - $QC_{2(i)}$) = (9/5 -8/8) = 4/5
- The eight software quality in consideration is Maintainability. Hence $QM_{(i)}$ = 8:8;
  $QM_{(f)}$ = 11:6
  $QM(I)$ is however ($QM_{(f)}$ - $QM_{(i)}$) = (11/6 – 8/8) = 5/6
- The ninth software quality is Testability with $QT_{(i)}$ = 8:8; $QT_{(f)}$ = 10:6
  $QT(I)$ = ($QT_{(f)}$ - $QT_{(i)}$) = (5/3 – 1) = 2/3
- The tenth software quality is Usability with $QU_{2(i)}$ = 8:8; $QU_{2(f)}$ = 11:5
  $QU_2(I)$ = ($QU_{2(f)}$ - $QU_{2(i)}$) = (11/5 – 8/8) = 6/5
- The eleventh software quality is Reliability with $QR_{(i)}$ = 8:8; $QR_{(f)}$ = 11:6
  $QR(I)$ = ($QR_{(f)}$ - $QR_{(i)}$) = (11/6 – 8/8) = 5/6
- The twelfth software quality is Efficiency with $QE_{(i)}$ = 8:8 , $QE_{(f)}$ = 8:4
  $QE(I)$ = ($QE_{(f)}$ - $QE_{(i)}$) = (2-1) = 1
- The thirteenth software quality is Integrity/Security with $QIS_{(i)}$ = 8:8; $QIS_{(f)}$ = 10:4
  $QIS(I)$ = ($QIS_{(f)}$ - $QIS_{(i)}$) = (5/2- 1) = 3/2

The overall improvement (QI) of the entire software system will therefore be gathered from the unit improvement of all the highlighted software qualities as calculated above;
It therefore means that;

$$QI = \sum \; [QU1(I) + QC1(I) + QI1(I) + QR(I) + QI2(I) + QP(I) + QC2\,(I) + QM(I) + QT(I) + QU2(I) + QR(I) + QE(I) + QIS(I)]$$

$$QI = \sum \; [\frac{4}{14} + \frac{7}{5} + \frac{3}{2} + \frac{9}{5} + 1 + \frac{2}{3} + \frac{9}{5} + \frac{5}{6} + \frac{2}{3} + \frac{6}{5} + \frac{5}{6} + 1 + \frac{3}{2}]$$

QI = 5 69/70 + 6 + 2 ½
QI = 14 (34/70)

Meaning that the force field analysis employed has proofed to be a vital tool, improving the current level of software qualities in the order of 14!. This goes to mean that if the software developers work on the supporting factors highlighted on the left side of the simulated force field model for each software quality, the overall productivity of the current level of software systems is bound to increase by fourteen (14) times.

## 14 Conclusion

The main objective of the research was to design a software quality improvement model. In this project, we highlighted 14 software qualities. On each software quality, the supporting and restraining factors were highlighted based on our field research. I have been able to suppress the constraints and reinforce the supporting factors using force field analysis concepts. However, my plight to improve each software feature and quality will remain a wishful thinking if the software developers and users do not adhere to the instructional procedures preached by the model designed from my analysis.

It should be clear from the model designed in this project that, if one uses a defined, reproducible process, one can always improve on the software product by removing the cause of repeatable problems. Nevertheless, we also need to be realistic. It is too difficult to develop software that would be free of all risk and constraints. However, it should be a cardinal sin to reproduce old mistakes. It is a duty of a software manager to always identify the chances of software improvements and constraints that limit the performance of a software package.

This will definitely assist the software developers to prevent, detect and react accordingly to the constraints that threaten the performance of a software package.

## 15 Recommendations

My recommendations would be based on the facts established in the analysis to improve each software quality. Hence on each software quality featured in this project, I therefore recommend the following;

1. To improve on **Understandability** feature of software, I recommend that;

    - The software system designed by the developer should be well- structured to accommodate new features.
    - The software developers should use software components authorized by the appropriate body e.g (ISO).

2. To improve on **Completeness** attribute in a software product, I recommend that;

    - The software system should be designed in models for easy access
    - The software developers (or vendors) should make the new software products compatible with old products and different computer configurations.
    - There should be data commonality in the specification of different vendor’s software components.

3. To enhance **Reusability** feature in a software system, I recommend that;

    - The software system should be self-descriptive
    - The software system should be compatible with different computer configurations.
    - The software package should be designed in modules for easy access.

4. To incorporate optimal **Integrability** feature in a software system, I recommend that;

- The software system should be made compatible with different vendor products.
- The software development process should be in modules to enhance incorporation of software additives.

5. To improve on **Portability** feature of a software system, I recommend that;

   - The software packages should be self-descriptive
   - The software system should be self- independent
   - The software system should be designed to perform very well irrespective of the hardware vendors.

6. To improve on **Consistency** feature in a software system, I recommend that;

   - The software system should be traceable and scalable with the old software package.
   - The software system should be designed to give the same service irrespective of the location
   - Data stored in one database should be replicable to another system irrespective of the software vendor.

7. To improve on **Maintainability** feature of a software system, I recommend that;

   - The software system should be made scalable and easy to upgrade
   - The software engineers should be proficient in validating the modified software systems.
   - The software engineers should be enlightened to know the aftermath of modifications on the software system

8. To improve on feature of **Testability** of a software product, I recommend that;

   - The software system should be accurately compatible with the software system
   - The software system should be designed in modules to enhance simplicity
   - The software system should be self-descriptive
   - The software developers should engage in using appropriate software development models.

9. In order to improve **Usability** feature of a software system, I recommend that;

   - The users of software systems should have optimal knowledge of its constituent's packages.
   - Users should be encouraged and motivated about using new technologies.

10. To make software system ***more Reliable***, I recommend that;

    - The software system should be designed and developed robustly to be able to continue where it stopped whenever there is a power outage without any loss of data.
    - The software system should be made to perform its tasks accurately
    - The developers should be given enough time to prepare and develop the software package. In fact, it has been observed that short time-to-market schedule degrades the quality of a software product.

11. Also, to enhance **Efficiency** of a software system, I recommend that;

    - There should be efficient use of system resources
    - The software system should be designed capable to increase its speed of production in a multi-user environment.

12. To improve on the **Security/Integrity** of a software system, I recommend that;

    - The software systems, especially the inventory software packages should possess an audit feature.
    - The software developers should incorporate strong and sophisticated security mechanisms such as user authentication, device authentication as well as Advanced Encryption key Standard (AES). These are to checkmate attacks and threats posed by hackers.

Finally, I wish to conclude this project saying that "***the race of software quality has no finish line***". Thus, for further quality improvement in Nigerian software development organizations, another important area opened for future research is on the proposition of force-field analysis on software agent activities and decisions. This would enable the software agents to be more independent, self-descriptive, social and autonomous.

## Competing Interests

Author has declared that no competing interests exist.

## References

[1] Royce WW. Managing the department of large software systems: Concepts and techniques. Proc. Wescon, August 1970. Also available in proceedings, ICSE 9, IEEE/ACM; 1987.

[2] Boehm B. Software risk management. IEEE Computer Society Press, CA; 1989.

[3] Capper Jones. Assessment and Control of Software Risk; Prentice Hall, ISBN 0-13-741406-4;711; 1994.

[4] June Verner. Interim progress report. National ICT, Australia; 2006.

[5] Force Field Analysis for Project Selection. (August, 2014); Available:www.brighthub.com/Forcefieldanalysis; www.portal.bright.com/

[6] Boehm BW. Risk management focuses the project manager's attentions on those portions of the project most likely to cause trouble and compromise participants' win conditions. IEEE Computer Society Press, New York; 1989.

[7] Franca Egbokhare. Analysis of human interactions in software development. Computer Science Section, Uniben' Library, Benin city, Nigeria; 2006.

[8] Boehm B, Bose P. A collaborative spiral software process model based on theory W. Proceedings, ICSP 3, IEEE, Reston; 1994.