

Software Quality Improvement Using Force-field Analysis

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 our plight to improve the performance of software systems, we began by seeking the opinions of software developers, having conducted personal interview with some of the developers; we suspected that some of them are being exaggerative about the processes of software development. This made us to change our 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 our 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 we decided to work with the information received from them.

In the plight of improving the current level of performance of the software systems, we 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, we have been able to establish a bench-mark that forms a foundational model for every software development project.

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.

Background of Study

In her article published in 1994, 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) 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.

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 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.

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 [5] 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 [3] 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.

104 **Aim and Objectives**

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

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

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

114 **Significance of Study**

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

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

124 **Scope of Study**

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

130 **Limitation of Research**

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

135 Also, many software users had their software systems developed from outside the country (e.g Banks,
136 Capital Trust Stock Exchange, Starcomms, TMS Travels et c.) This also deprived us from reaching their

137 software developers to actually study and access the software development processes from their
138 respective developers.

139 **Methodology of Research**

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

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

- 149 i. Know the current level of performance of the existing software systems used in Nigeria
- 150 ii. Identifying the pro's and con's in each software development project
- 151 iii. Identifying the actions involved in software development process and the resulting reactions or
152 consequences associated to each action on every software development project
- 153 iv. To compare the ideal situations and realities in each software development process.
- 154 v. Attempt to bridge the gap between the user's priority and developer's views.

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

- 158 i. Software Development Organization
- 159 ii. Educational Institutions
- 160 iii. Banks
- 161 iv. Research Institutes
- 162 v. State Government Parastatals

163

164 The heterogeneous sample indulged in this research consists of the following;

- 165 i. Software team leaders of Software Development Organizations
- 166 ii. Software Developers
- 167 iii. Users of Software Systems

168 **Survey Instruments**

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

- 170 i. Questionnaires

171 ii. Key Informant Interview

172 Questionnaires

173 180 structured questionnaires containing questions related to some selected software quality attributes
174 were administered to users of software systems. For the purpose of this study, we grouped users into
175 two main categories:

176 i. Power Users

177 ii. Common or End Users

178 **Power Users:**These are the personnel who make use of the software systems to render services to the
179 populates.

180 **Common or End Users:**These are the receiving audience who make use of the services being rendered
181 by the power users.

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

192 Firstly, to affirm the reasons of this research, we started with the common users. About 180 structured
193 questionnaires on the software qualities of our interest (about 14 of them) were administered, of which
194 150 were usable for the analysis. The first part of the questionnaires sought information about
195 demography of the users in order to know their level of relevance with IT tools. The second part sought
196 information about the type (and purpose) of the software systems while the third part sought opinions
197 from the respondents on their software quality assessment. The following sections constitute the
198 analysis of the questionnaires.

199 Demographic Information

200 Academic/Professional Qualification(s)

	Frequency	Percentage
PRIMARY	1	0.7
SECONDARY	78	52.0
POST-SECONDARY	71	47.3
TOTAL	150	100

201 *Table 1: Showing the Academic profile of the Examined Common Users*

202 **SEX**

GENDER	FREQUENCY	PERCENTAGE
MALE	102	68.0
FEMALE	48	32.0
TOTAL	150	100.0

203 *Table 2: Displaying the Gender Status of the Common Users*

204 **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

205 *Table 3: Showing the age-range of the Common Users*

206 **JOB EXPERIENCE**

YEARS OF EXPERIENCE	FREQUENCY	PERCENTAGE
1-5 Years	51	34
6-10	78	52
Above 10 Years	28	14
TOTAL	150	100

207 *Table 4: Displaying the Years of experience of the Examined Common Users*

208

209 **EMPLOYMENT STATUS**

Job Status	FREQUENCY	PERCENTAGE
Employer	30	20.0
Employee	120	80.0
TOTAL	150	100

210 *Table 5: Illustrating the Job Status of the Common Users*

211 Table 1 to Table 5 reflects the demographic profile of the common users. Table 1 enumerates the
 212 academic profile of the common users interviewed; the information obtained from the Table1 reflects

that only 1 personnel which constitutes just 0.7% has a primary school certificates, about 78% 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.

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
Maintenability	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

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.

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 below:

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
Maintenability	121	29
Testability	123	27
Usability	128	22
Reliability	133	17
Efficiency	129	21
Security/Maintenance	138	12

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 looked into.

Questionnaires containing three parts were administered by the researcher to these software users. Based on the demand to reach optimally from the common users initially interviewed and constraints attached to each software quality and feature in consideration, series of questions were structured into questionnaires. The set of questionnaires were channeled towards common user's view on the software systems. The reason why we have to focus on users is because the developers are being exaggerative about their mode of development and as such, are reluctant to relay information about the weaken aspects of software development. We felt 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 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 with the software products.

The first part of the questionnaire sought information about demographic information 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 system while the third part sought the opinions of the respondents on their software qualities. A questionnaire was designated to one organization. In other words, the initial aim was to carry out research on sixteen companies but eventually only fourteen of them were usable for the analysis.

The following section reflects the analysis of the data gathered during our second phase of field research.

ANALYSIS OF SOFTWARE QUALITIES PRESENT IN THE SOFTWARE SYSTEMS USED BY RESPONDENTS

Table 8: The following 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.

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				

280 **Notable Keys:**

281 **EX = Excellent**

282 **V.G = Very Good**

283 **% of = Percentage of**

284

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

INDICATION OF INTEREST IN OPTIMIZING SOFTWARE 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

287

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

	Frequency	Percentage
YES	8	57.1
NO	6	42.9
TOTAL	14	100.0

290 **Table 10: Signifying users who prefer a better software packages**

291 The above estimation shows that 8 (57.1%) of the respondents would prefer a better software package,
292 while 6 (42.9%) do not prefer a better software package. This implies that majority of software users
293 would prefer a better software package.

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

	Frequency	Percentage ²⁹⁶
YES	1	7.1 ²⁹⁷
NO	13	92.9
TOTAL	14	100.0 ²⁹⁸

299 ***Table 11: Indicating the Power User Company having interest in changing their Software Developers***

300

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

303 **Key Informant Interviews**

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

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

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

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

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

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

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

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

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, we are using the common users' view and analysis as the basis for the research. We later proceeded to identifying the supportive factors and problems associated with each software quality. Using the requirement negotiation processes embedded in force field analysis, we were able to establish a bench-mark that forms a foundational model for every software development project.

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.

Result and Interpretation

In the last section, 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, we gave our 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.

We 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 talk 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 motivation needs and desires and the software developers' priority, using the requirement negotiation process embedded in win-win software development model, we can shed away the unprofessional practices as well as unethical pattern of usage of the common users. Also, using this model we 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.

Details of win-win Spiral Software Development Model

The developed Win-win Spiral Model [4] uses the theory W (win-win) approach [5] to converge on a system's next-level objectives, constraints, and alternatives. In figure 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.

The win-win Spiral Model of Software Development

387

389

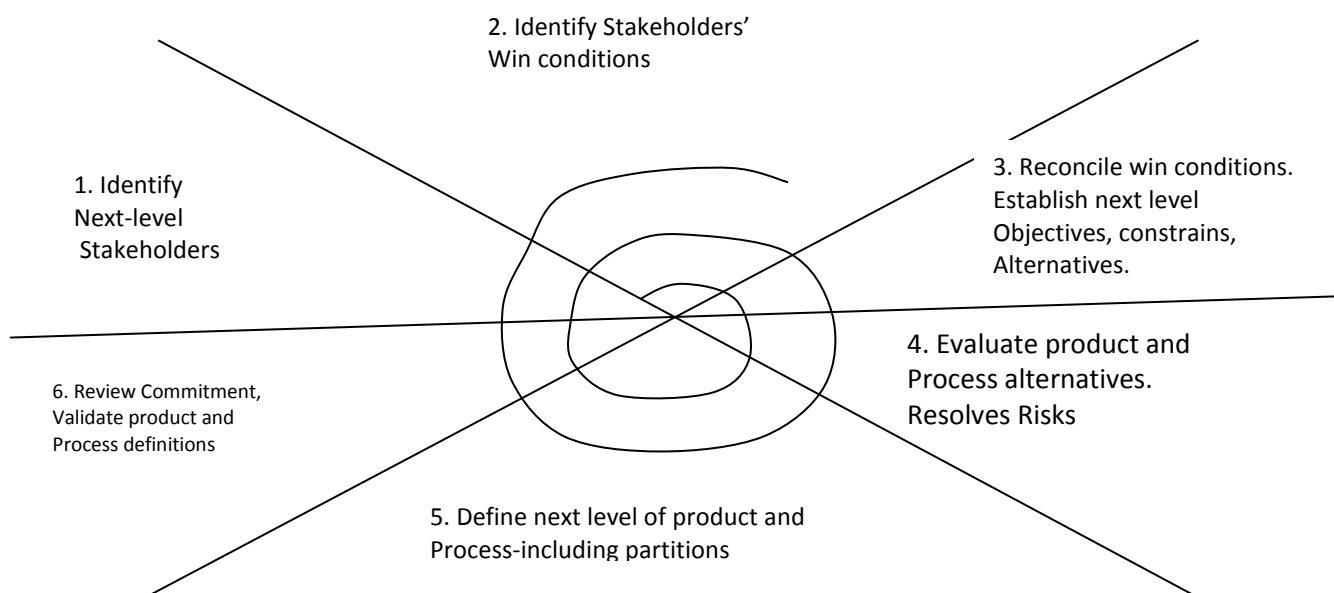


Figure 1: The win-win Spiral Model

390 **Determine Objectives:** Identify the system life-cycle stakeholders and their win conditions. Establish
391 initial system boundaries and external interfaces.

392 **Determine Constraints:**Determine the conditions under which the system would produce win-lose or
393 lose-lose outcomes for some stakeholders.

394 **Identify and Evaluate Alternatives:**Solicit suggestions from stakeholders.

395 Evaluate them with respect to stakeholders' win conditions. Synthesize and negotiate candidate win-win
396 alternatives. Analyze, Assess, and resolve win-lose or lose-lose risks.

397 Record Commitments and areas to be left flexible, in the project's design record and life cycle plans.

398 **Cycle Through the Spiral:**Elaborate win conditions, screen alternatives, resolve risks, accumulate
399 appropriate commitments, and develop and execute downstream plans.

400

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

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

416 **Resolving win Conditions- Establish next Level Objectives, Constraints and Alternatives**

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

422

Incorporating Force Field Analysis and its Concepts

Force Field Analysis was a concept firstly pioneered by a Social Psychologist, Lewin Kurt [6]. 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.

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. According 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.

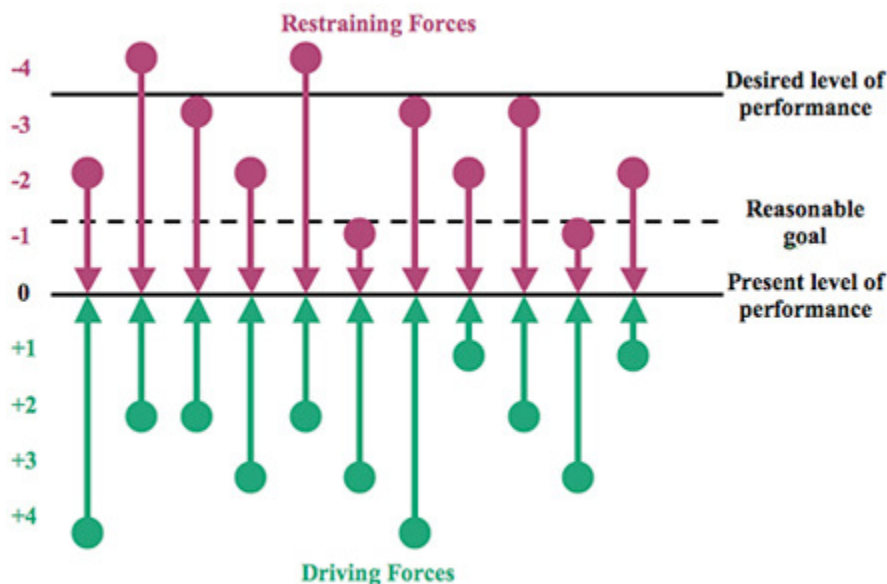


Figure 2: Lewin Kurt's Model on Force-field Analysis

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.

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.

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 Figure 3 below;

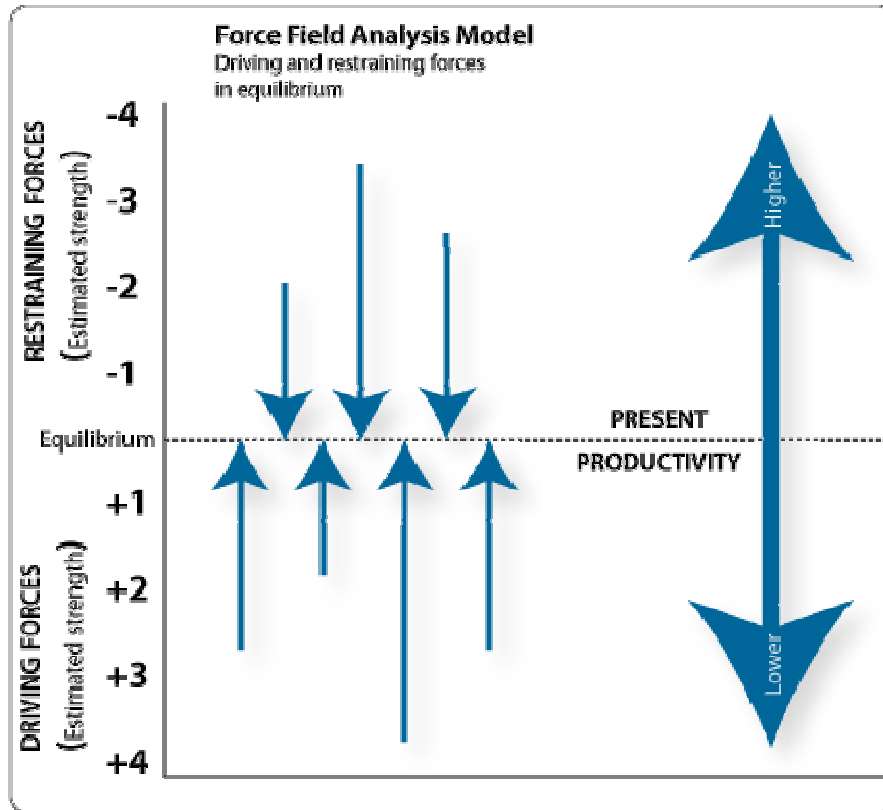


Figure 3: A Scientific Interpretation of Lewin Kurt's Model

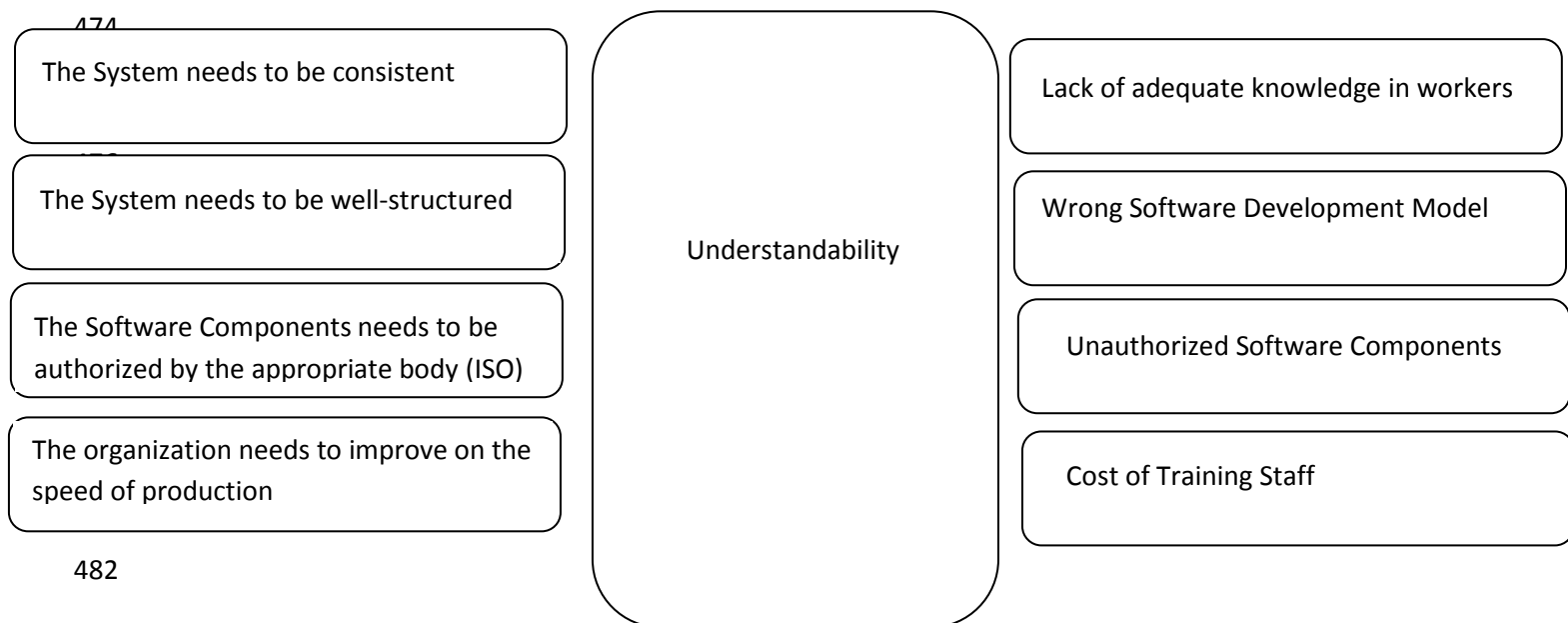
This equilibrium, or present level of productivity, can be raised or lowered by changes in the relationship between the driving and restraining forces. In order to relate the concept of force field analysis to the on-going project, my procedure demands looking into those software features and qualities 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

UNDER PEER REVIEW

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)



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

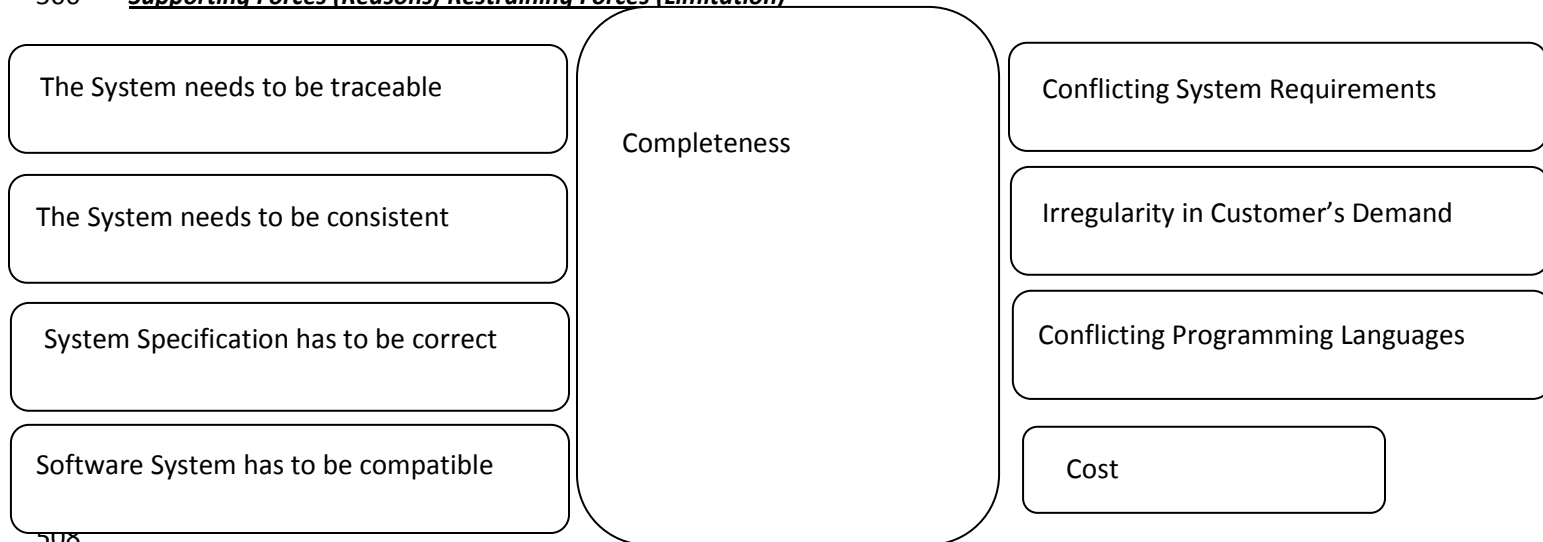
- ❖ Adequate training for staff increases cost but also enhances knowledge of staff (+1,-1, **strength to the restraining forces but also take care of the limitation of unskilled workers**)
- ❖ Improve speed of production and reduce work overtime (+1, **strengthen to the supporting forces**)
- ❖ Use 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**)

495 **The Result**

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

499 Second on the list is Completeness

500 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



508
509 If we are to assign eight points to both the supporting forces and the restraining forces respectively. The
510 analysis goes as follows:

- 511 ❖ It is costly to make a complete software product **(+1, reinforcing the restraining forces)**
- 512 ❖ Making use of win-win software development model with the embedded software requirement
513 negotiation process in the model will checkmate irregularities in customer's demands **(-1,**
514 **reducing the strength of the restraining forces)**
- 515 ❖ Making use of software component authorized by international standard organization (ISO) will
516 take care of different vendor specification because they are all authorized by the same
517 authorizing body and as such, will accommodate or permit the same range of functional
518 requirements **(+1,-1 reducing the strength of the restraining forces and increasing the strength of the**
519 **supporting forces)**
- 520 ❖ Making use of a software development model that will accommodate a comprehensive
521 deliverable for each stage of software development will help refine the needs and the
522 requirements (i.e. specification) accurately and will also enhance consistency, **(+2, reinforcing the**
523 **supporting forces)**
- 524 ❖ Making use of a compatible programming languages or conventional programming languages
525 that has extensive library facilities will help incorporate old programming languages **e.g Fortran**
526 **66-to-77-to Fortran 7x et.c(-1, reducing the restraining forces)**

527

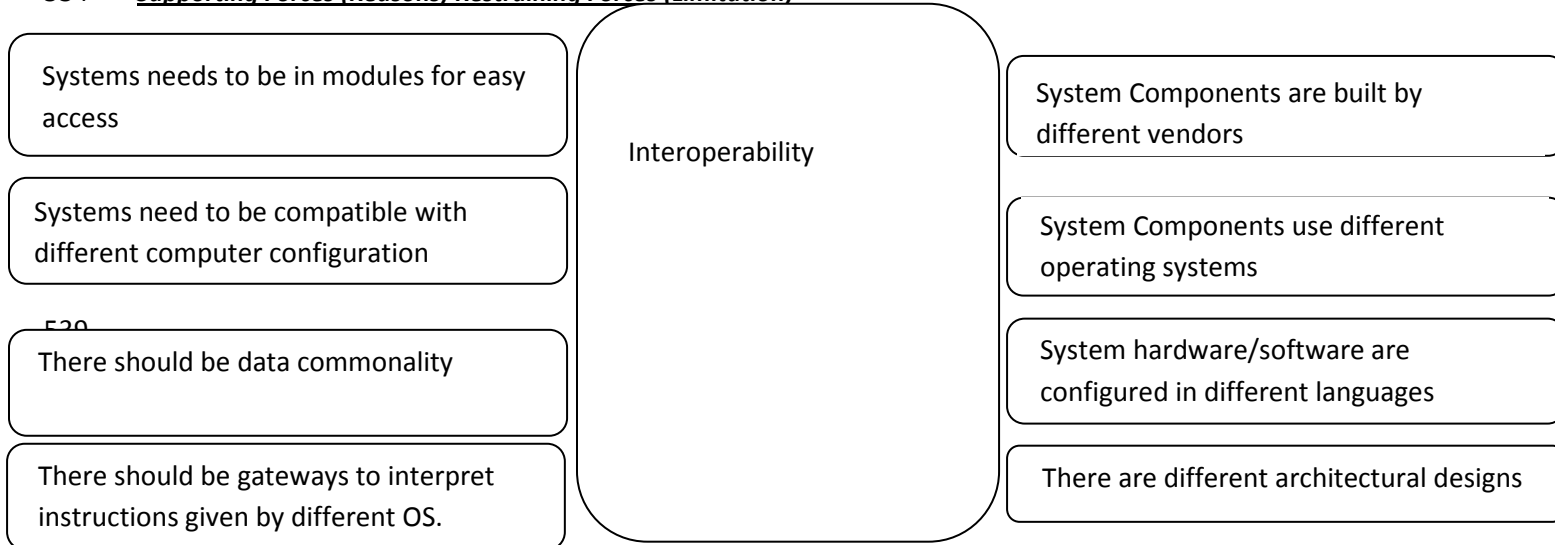
528 **The Result**

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

532

533 Third on the list is **interoperability**

534 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



543

544 **The Analysis**

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

- 547 ❖ Making use of a software packages using the same or upgrade operating systems e.g Window
548 2003-to-2007-to-Vista, **(-1, reducing the strength of the restraining forces)**
- 549 ❖ Installation of Gateway software for the conversion and translation of various programming
550 languages and operating systems into comprehensive and understandable ones **(+1,-1, reducing**
551 **the strength of the restraining forces and increasing the strength of the supporting)**
- 552 ❖ Making use of a compatible programming languages or conventional programming languages
553 that has extensive library facilities will help incorporate old programming languages e.gFortran
554 66-to-Fortran 77-to-Fortran 7x et c., **(-1, reducing the strength of the restraining forces)**
- 555 ❖ Making use of software component authorized by international standard organization (ISO) will
556 take care of different vendor specification because they are all authorized by the same
557 authorizing body and as such, will accommodate or permit the same range of functional

558 requirements (-1,+1, reducing the strength of the restraining forces and increasing the strength of the
559 supporting forces)

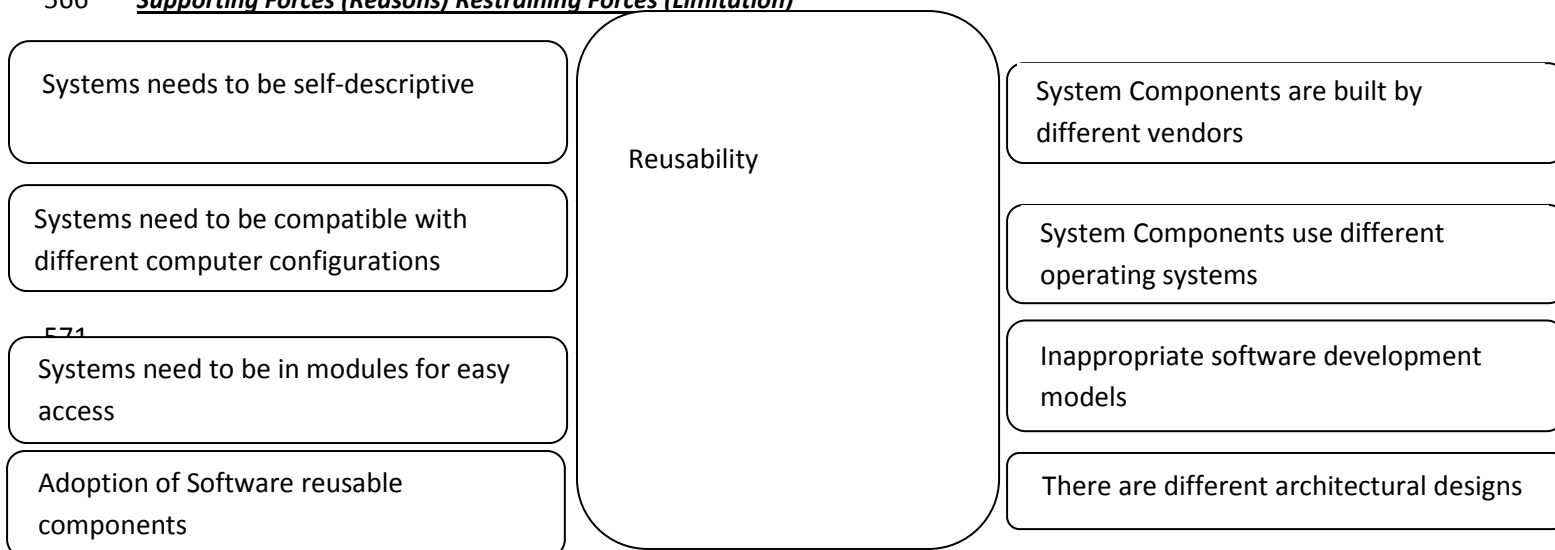
560 *The Result*

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

564

565 Fourth on the list is **Reusability**

566 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



575

576 *The Analysis*

- 577 ➤ Making use of intelligent software agents (+2, reinforcing the supporting forces)
- 578 ➤ Making use of software components authorized by international standard organization (ISO) will
579 take care of different vendor specification because they are all authorized by the same
580 authorizing body and as such, will accommodate or permit the same range of functional
581 requirements (-1,+1, reducing the strength of the restraining forces and increasing the strength of the
582 supporting forces)
- 583 ➤ Making use of a software packages using the same or upgradable operating systems e.g.
584 Windows 2003-to-2007-to-Vista (-1, reducing the strength of the restraining forces)
- 585 ➤ Installation of Gateway softwares for the conversion and translation of various programming
586 languages and operating systems into comprehensive and understandable ones (-1,+1, reducing
587 the strength of the restraining forces and increasing the strength of the supporting forces)

588 ➤ Making use of a win-win software development model with appropriate delivery for each stage
589 of the developmental process will make the system accessible in modules **(+1, reinforcing the**
590 **supporting forces)**

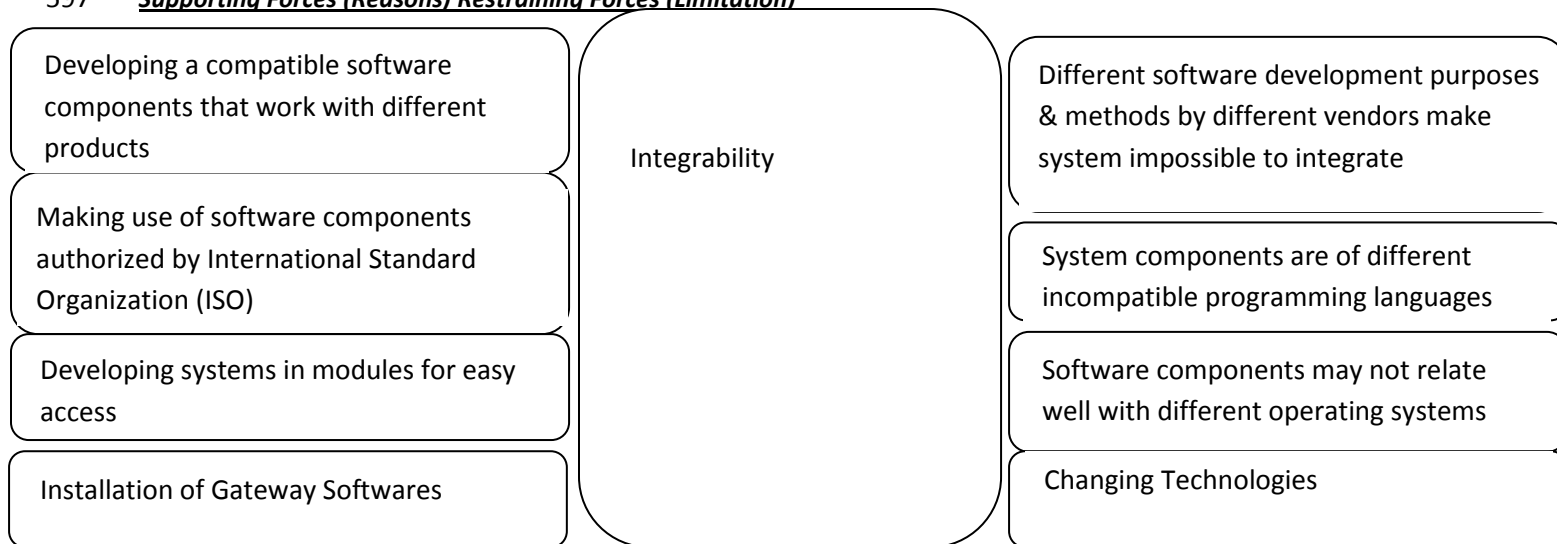
591 *The Result*

592 In the end, the ratio between the supporting forces and restraining forces on software reusability has
593 changed through this analysis from **8:8** to **13:5** in favor of the supporting forces. *Hence, there is a*
594 *considerable improvement.*

595

596 Fifth on the list is **Integrability**

597 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



606

607 *The Analysis*

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

610 ➤ Making use of software components authorized by International Standard Organization
611 (ISO) will take care of different vendor specification because they are all authorized by
612 the same authorizing body and as such, will accommodate or permit the same range of
613 functional requirements **(-1, reducing the strength of restraining forces)**

614 ➤ Installation of Gateway Software **(-1,+1, reducing the strength of the restraining forces and**
615 **increasing the strength of the supporting forces)**

616 ➤ Making use of an updated software development method and CASE tools e.g Web-
617 centric software development method **(-1, reducing the strength of the restraining forces)**

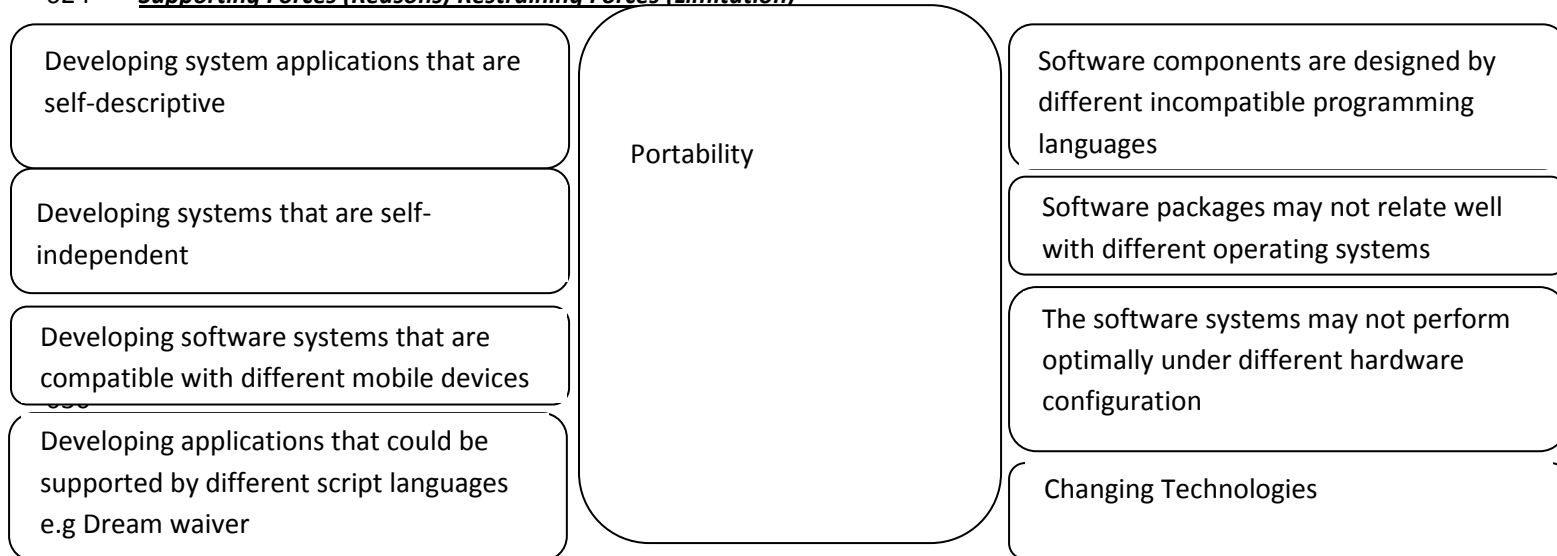
618 The Result

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

622

623 Sixth on the list is **Portability**

624 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



633

634 *The Analysis*

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

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

646

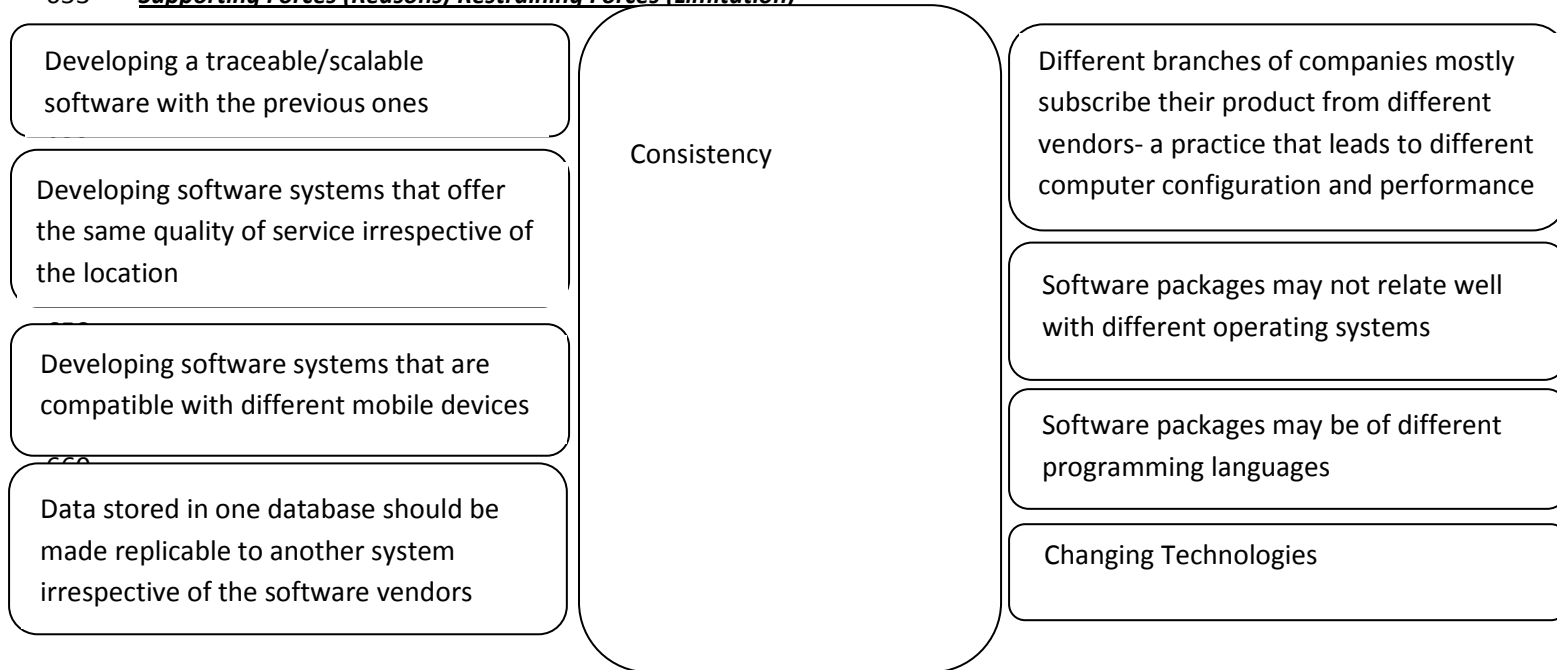
647 The Result

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

651

652 The seventh quality is **Consistency**

653 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



663

664 *The Analysis*

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

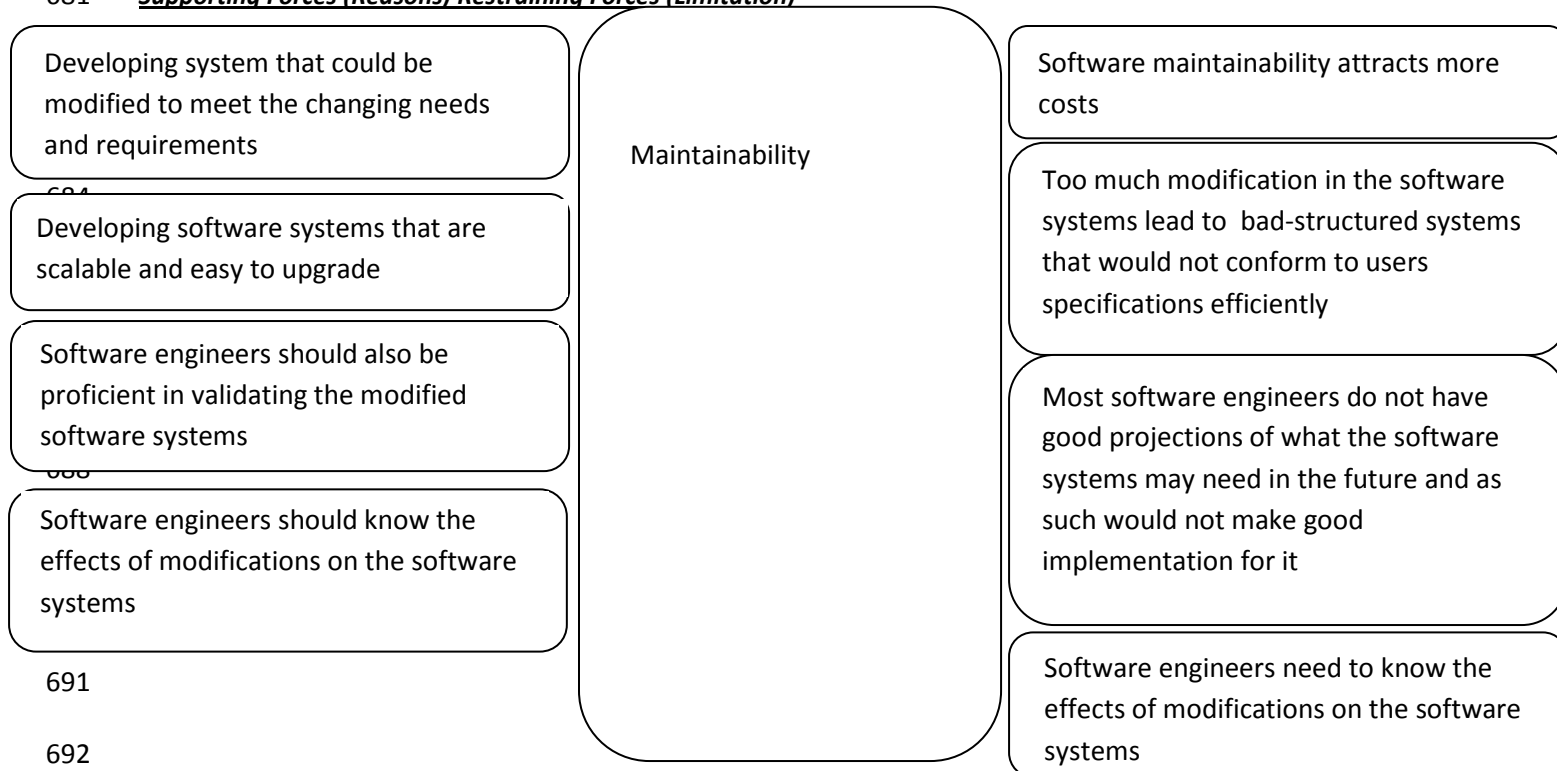
- 667 ➤ Making use of a conventional programming languages that could be easily be upgraded
668 e.g. C-to - C+ - to - C# (-1, reducing the strength of the limiting forces)
- 669 ➤ Installation of gateway software and appropriate compilers (-1, reducing the strength of the
670 limiting forces)
- 671 ➤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

676 The Result

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

680 The Eighth Quality is **Maintainability**

681 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



693 *The Analysis*

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

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

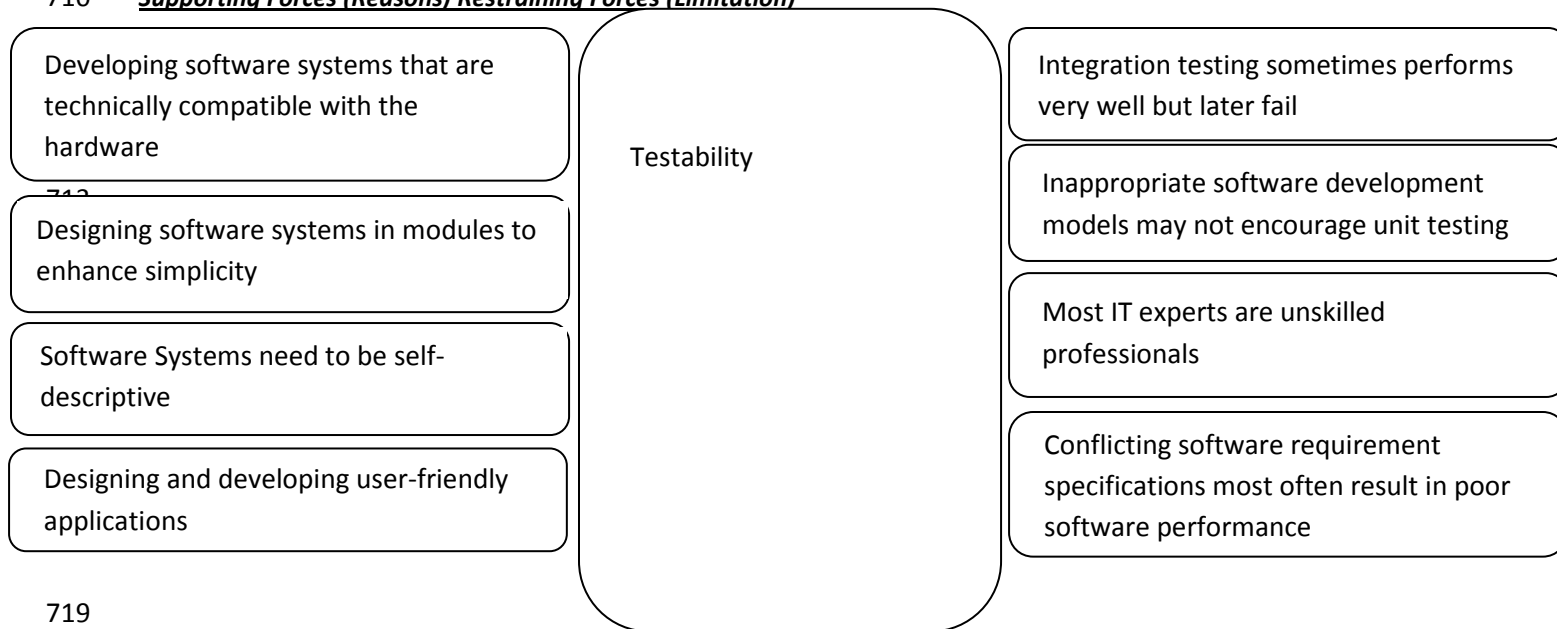
704

705 The Result

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

709 The Ninth Quality is **Testability**

710 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



721 *The Analysis*

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

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

733

UNDER PEER REVIEW

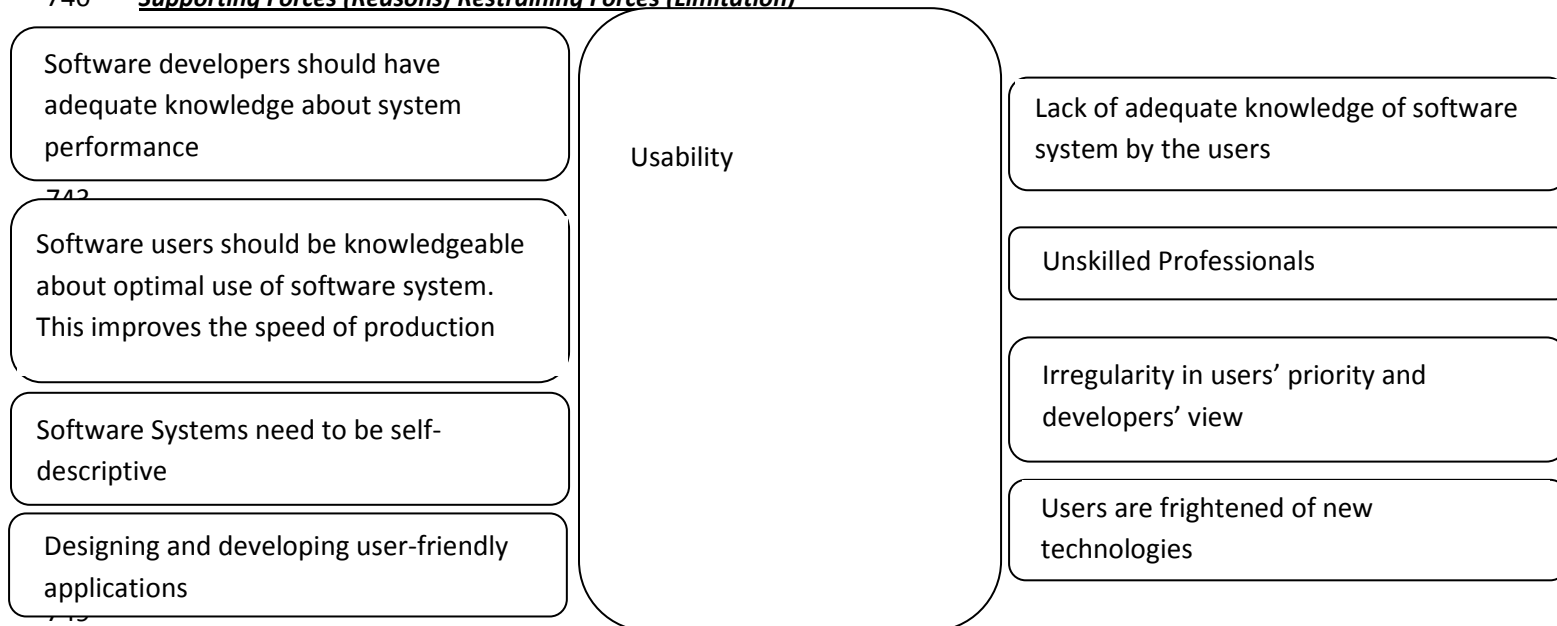
734 The Result

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

738

739 The Tenth quality is **Usability**

740 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



750

751

752 *The Analysis*

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

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

UNDER PEER REVIEW

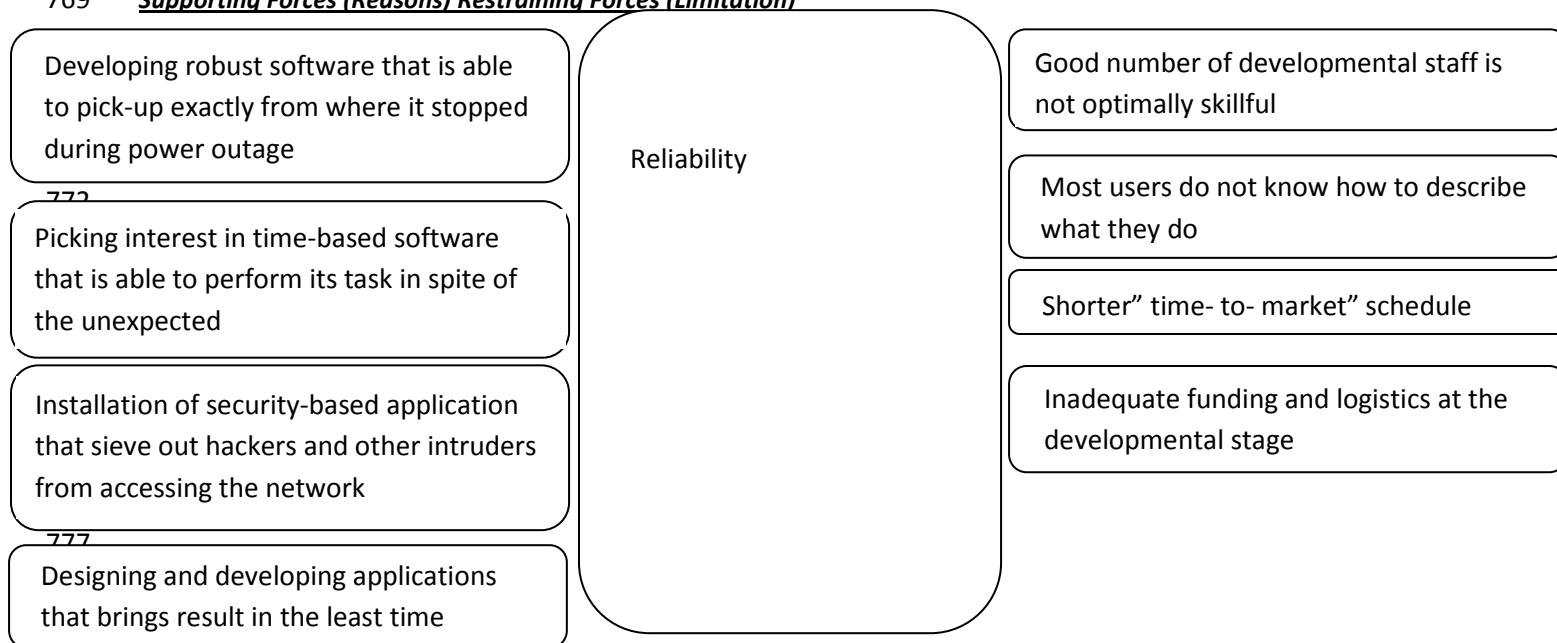
763 The Result

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

767

768 The Eleventh quality is **Reliability**

769 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



779

780 *The Analysis*

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

- 783 ➤ Adequate training for software developers and prospective users (+1,-1, raising the strength
784 of the supporting forces but also reducing the strength of the restraining forces)
- 785 ➤ Making use of deliverables that will give good account of each stage of developmental
786 process will enhance accuracy in the software performance (+1, reinforcing the supporting
787 forces)
- 788 ➤ Engage in a win-win software developmental process that facilitate software
789 requirement negotiation activities (-1, reducing the strength of the limiting forces)

UNDER PEER REVIEW

790 ➤ Incorporation and installation of security-based applications that protect hackers and
791 other intruders would help sieve out instability in software performance (+1, reinforcing
792 the supporting forces)

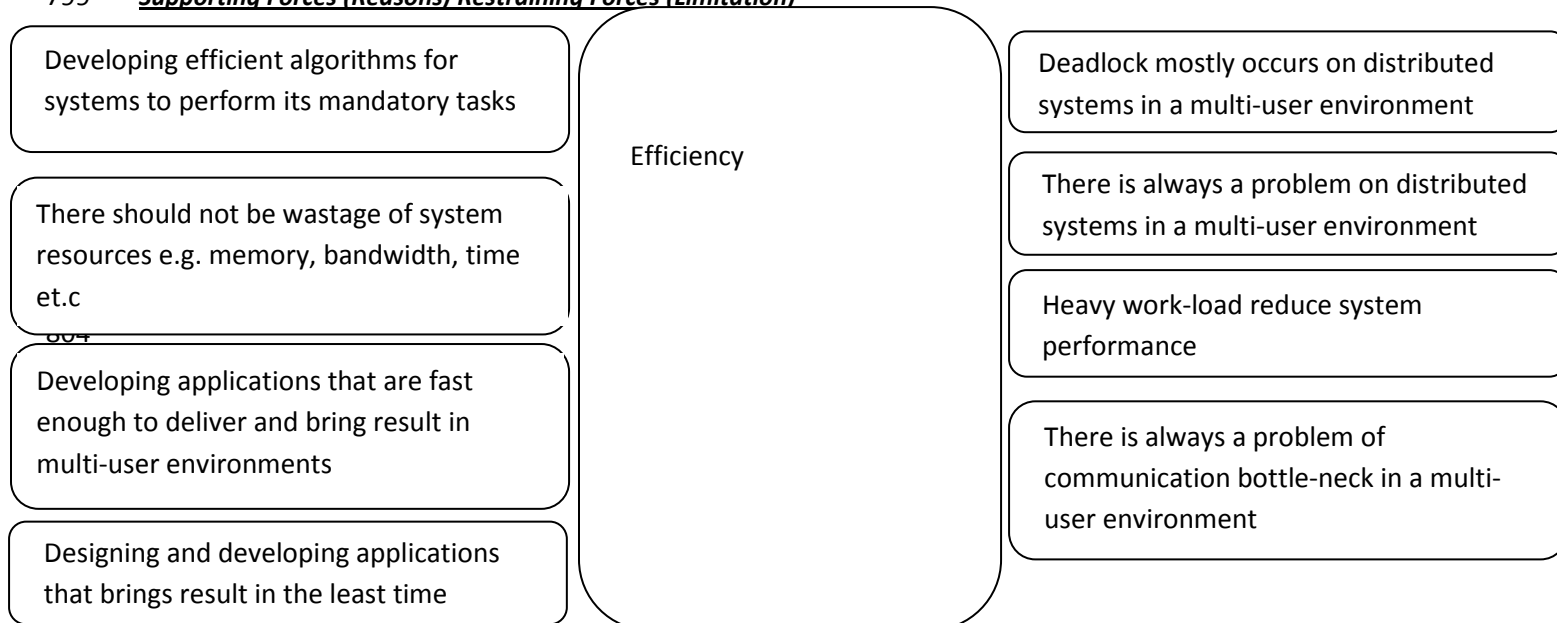
793 The Result

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

797

798 The twelfth quality is **Efficiency**

799 **Supporting Forces (Reasons) Restraining Forces (Limitation)**



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

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

818

UNDER PEER REVIEW

819 The Result

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

823

824 The Thirteenth quality is **Integrity/Security**

825 **Supporting Forces (Reasons) Restraining Forces (Limitation)**

Designing, developing and
implementing a protective access
control for systems in a multi-user
environment

829

Incorporating inventory software systems
that implement audit feature

831

Implementing systems that are well
protected against virus and hackers

833

Designing encryption keys that are
difficult to crack by hackers

835

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

836

837 *The Analysis*

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

- 840 ➤ Developing anti-virus to protect hackers' attack (-1,+1, **reducing the strength of the restraining**
841 **forces and increasing the strength of the supporting forces)**
- 842 ➤ Making use of a wireless technology like WLAN with good security features reduce
843 unnecessary interruption from hackers (+1, **reinforcing the supporting forces)**
- 844 ➤ Software users should be assigned a username and a password only to legitimate users
845 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.**

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; $QU_{1(i)}$, after the analysis $QU_{2(f)}$, and the resultant improvement is represented as $QU_{1(l)}$.

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	$QU_{1(i)}$	$QU_{1(f)}$	$QU_{1(l)}$
2	Completeness	$QC_{1(i)}$	$QC_{1(f)}$	$QC_{1(l)}$
3	Interoperability	$QI_{1(i)}$	$QI_{1(f)}$	$QI_{1(l)}$
4	Reusability	$QR_{(i)}$	$QR_{(f)}$	$QR_{(l)}$
5	Integrability	$QI_{2(i)}$	$QI_{2(f)}$	$QI_{2(l)}$
6	Portability	$QP_{(i)}$	$QP_{(f)}$	$QP_{(l)}$
7	Consistency	$QC_{2(i)}$	$QC_{2(f)}$	$QC_{2(l)}$
8	Maintainability	$QM_{(i)}$	$QM_{(f)}$	$QM_{(l)}$
9	Testability	$QT_{(i)}$	$QT_{(f)}$	$QT_{(l)}$
10	Usability	$QU_{2(i)}$	$QU_{2(f)}$	$QU_{2(l)}$
11	Reliability	$QR_{(i)}$	$QR_{(f)}$	$QR_{(l)}$
12	Efficiency	$QE_{(i)}$	$QE_{(f)}$	$QE_{(l)}$
13	Integrity/Security	$QIS_{(i)}$	$QIS_{(f)}$	$QIS_{(l)}$

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 $QU_{1(f)} = 10:7$

- 866 Therefore, $QU_1(I)$ is however $(QU_{1(ff)} - QU_{1(ii)}) = (10/7 - 8/8) = 4/14$
 867
- 868 ➤ Again, for the second software quality Completeness $QC_{1(ii)} = 8:8$ and $QC_{1(ff)} = 12:5$
 869 $QC_1(I) = QC_{1(ff)} - QC_{1(ii)} = (12/5 - 8/8) = 7/5$
 870
- 871 ➤ For the third software quality, Interoperability, $QI_{1(ii)} = 8:8$; $QI_{1(ff)} = 10:4$
 872 $QI_1(I) = (QI_{1(ff)} - QI_{1(ii)}) = (5/2 - 1) = 3/2$
 873
- 874 ➤ For the fourth software quality, Reusability, $QR_{(ii)} = 8:8$; $QR_{(ff)} = 13:5$
 875 $QR(I) = (QR_{(ff)} - QR_{(ii)}) = (13/5 - 8/8) = 9/5$
 876
- 877 ➤ The fifth software quality is Integrability; $QI_{2(ii)} = 8:8$; $QI_{2(ff)} = 10:5$
 878 $QI_2(I) = (QI_{2(ff)} - QI_{2(ii)}) = (2 - 1) = 1$
 879
- 880 ➤ For the sixth software quality; Portability, $QP_{(ii)} = 8:8$, $QP_{(ff)} = 10:6$
 881 $QP(I) = (QP_{(ff)} - QP_{(ii)}) = (5/3 - 1) = 2/3$
 882
- 883 ➤ The seventh software quality is Consistency; and $QC_{2(ii)} = 8:8$, $QC_{2(ff)} = 9:5$
 884 $QC_2(I) = (QC_{2(ff)} - QC_{2(ii)}) = (9/5 - 8/8) = 4/5$
 885
- 886 ➤ The eight software quality in consideration is Maintainability. Hence $QM_{(ii)} = 8:8$;
 887 $QM_{(ff)} = 11:6$
 888 $QM(I)$ is however $(QM_{(ff)} - QM_{(ii)}) = (11/6 - 8/8) = 5/6$
 889
- 890 ➤ The ninth software quality is Testability with $QT_{(ii)} = 8:8$; $QT_{(ff)} = 10:6$
 891 $QT(I) = (QT_{(ff)} - QT_{(ii)}) = (5/3 - 1) = 2/3$
 892
- 893 ➤ The tenth software quality is Usability with $QU_{2(ii)} = 8:8$; $QU_{2(ff)} = 11:5$
 894 $QU_2(I) = (QU_{2(ff)} - QU_{2(ii)}) = (11/5 - 8/8) = 6/5$
 895
- 896 ➤ The eleventh software quality is Reliability with $QR_{(ii)} = 8:8$; $QR_{(ff)} = 11:6$
 897 $QR(I) = (QR_{(ff)} - QR_{(ii)}) = (11/6 - 8/8) = 5/6$
 898
- 899 ➤ The twelfth software quality is Efficiency with $QE_{(ii)} = 8:8$, $QE_{(ff)} = 8:4$
 900 $QE(I) = (QE_{(ff)} - QE_{(ii)}) = (2 - 1) = 1$

901

902 ➤ The thirteenth software quality is Integrity/Security with $QIS_{(ij)} = 8:8$; $QIS_{(ff)} = 10:4$
 903 $QIS(I) = (QIS_{(ff)} - QIS_{(ij)}) = (5/2 - 1) = 3/2$
 904

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

907 It therefore means that;

908 $QI = \sum [QU1(I) + QC1(I) + QI1(I) + QR(I) + QI2(I) + QP(I) + QC2(I) + QM(I) + QT(I) +$
 909 $QU2(I) + QR(I) + QE(I) + QIS(I)]$

910 $QI = \sum \left[\frac{4}{14} + \frac{7}{5} + \frac{3}{2} + \frac{9}{5} + 1 + \frac{2}{3} + \frac{9}{5} + \frac{5}{6} + \frac{2}{3} + \frac{6}{5} + \frac{5}{6} + 1 + \frac{3}{2} \right]$

911 $QI = 5 \frac{69}{70} + 6 + 2 \frac{1}{2}$

912 $QI = 14 \left(\frac{34}{70} \right)$

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

918 Conclusion

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

926

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

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

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.

Reference(s)

[1] Royce (1970) W.W. Royce, “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] Franca Egbokhare (2006), “Analysis of Human Interactions in Software Development” Computer Science Section, Uniben’ Library, Benin city, Nigeria.

[4] Boehm B. and P. Bose (1994), “A Collaborative Spiral Software Process Model Based on Theory W”, Proceedings, ICSP 3, IEEE, Reston.

[5] Boehm B. W. (1989) “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.

[6] Force Field Analysis for Project Selection; Available at, www.brighthub.com/Forcefieldanalysis . Also available at www.portal.bright.com/