The status of Object-oriented Software Reuse and Reusability Assessment in the Kenyan Software Engineering Industry

Sammy Olive Nyasente*, Prof. Waweru Mwangi, Dr. Stephen Kimani

School of Computing and IT, Jomo Kenyatta University of Agriculture and Technology, PO Box 62000-00200 Nairobi Kenya

*Email of corresponding author: samqolive@yahoo.com

Abstract

There are published claims of widespread ad-hoc reuse within the software Engineering industry—a situation that has caused organizations not to gain optimal benefits from reuse. The general impression created by literature is that, software developers hardly consider the concept measurement as a way of assessing reusability of developed software, thus the resulting software lack adequate reusability. This result to a common conclusion that, the software Engineering industry is still grappling with software development challenges that reuse is intended to solve. The purpose of this paper is to provide empirical evidence on the status of reuse and reusability assessment, which should form a basis for addressing the problems that hinder effective reuse. This paper reports the findings of an empirical study that surveyed software developers who had knowledge in OO software development. From the analysis of fifty-four (54) valid responses, the study establishes the status of reuse and reusability assessment, as well as the perceptions and awareness of OO developers on the concept of software measurement, with regards to software quality. Based on the findings of the survey, we give recommendations on how organizations can improve the reuse practice.

Keywords: Software reuse, Software reusability, Software Metrics, Software Measurement

1. Introduction

The concept of software reuse stems from the fact that many software systems that are often engineered contain similar or identical components (Sametinger, 1997). This means that most software systems have similarity in functionality, design, code etc. It then follows that parts of existing software—such as requirements documents, designs, and code; can be used in building new software systems. According to a number of literatures, reuse is capable of resolving issues related to software quality, cost, and productivity (Sametinger, 1997; Frakes & Kang, 2005; Sommerville, 2011; Hristov, Hummel, Huq, & Janjic, 2012). This notwithstanding, reuse faces numerous challenges and lacks adoption by practitioners (Hristov et al., 2012). The major impediments to successful reuse revolve around the issue of reusability (Hristov et al., 2012; Sametinger, 1997). According to Sametinger, most of the existing software has little or no reusability. Frakes and Kang (2005), describe Reusability as a property that indicates the probability of reusing any software asset. According to (Nyasente, Mwangi, & Kimani, 2014), low levels of reusability diminish the chances of a software component being reused; therefore, efforts should converge at developing components with adequate reusability—in order to achieve effective reuse.

Although Object Oriented Software Development (OOSD) is capable of improving reusability, optimal reusability could be achieved through measurement (Nyasente et al., 2014). Software measurement enables software developers to objectively assess the status of different aspects of software products' quality—thus providing a basis for improvement (Chawla & Nath, 2013; Pressman, 2005).

This paper presents the findings of our initial inquiry into the use of metrics in reusability assessment of Object-oriented (OO) software. That is, the paper reports the results of a study that surveyed software developers who had experience in OO software design and development. Our long term goal is to develop a reusability assessment tool—based on the reusability measurement model that we presented in (Nyasente et al., 2014). We believe that understanding the current industry practice with regards to OO reusability assessment is requisite in achieving this goal—as there is little empirical research that studies the perceptions of developers towards reusability assessment, as well as factors that prevent them from assessing reusability. The key objectives of the work described herein are:

- To understand the manner in which reuse is conducted, as well as examine its efficacy as-is.
- To identify the challenges and opportunities for improving reuse.
- To determine the level of awareness and perceptions of OO developers with regards to software metrics and reusability assessment.

- To establish the methodologies deployed by developers in reusability assessment, as well as examine the efficacy of those methodologies.

The survey also covered a number of aspects that we believe are related to software reuse. Most importantly, the status and effect of technology use to support software design and development were investigated.

2. Literature Review

2.1. Reusability Assessment

Software measurement is a key aspect in good software engineering practice (Farooq, Quadri, & Ahmad, 2011). According to (Pressman, 2005), measurement is the only real way of determining the state of quality aspects of software being developed. Measurement can help developers to make specific software characteristics more visible (Farooq et al., 2011). According to Farooq et al., Measurement encompasses quantitative evaluations which use metrics and measures that can be used to determine attainment of numerical quality goals.

The claim by (Sametinger, 1997) that, most existing software has little or no reusability is an indication that developers hardly measure reusability when developing software. A similar view is held by (Hristov et al., 2012) when they state that, one of the impediments preventing efficient and effective reuse is the difficulty of determining artifacts that are best suited to solve a particular problem in a given context and how easy it will be to reuse them there. The authors further claim that, no framework that structures existing reusability metrics in a way that is easy to use can be found in literature. It is on this premise that they propose a metrics-based reusability assessment framework for identifying reusable components in ad-hoc reuse scenarios. Nyasente et al., (2014) also propose a metrics-based reusability assessment framework for OO software. The authors claim that, no effective framework for assessing the reusability of OO software exists in literature. This is in spite of the fact that various studies (Chidamber & Kemerer, 1994; Gill & Sikka, 2011; Chawla & Nath, 2013; Dubey & Rana, 2010), present OO metrics that can be used in assessing OO reusability. Other metrics-based frameworks for reusability assessment are presented in (Caldiera & Basili, 1991; Washizaki, Yamamoto, & Fukazawa, 2003; AL-Badareen, Selamat, Jabar, Din, & Turaev, 2010; Ilyas & Abbas, 2013).

2.2. The Nature of Software Reuse

The practice of reuse is as old as programming itself (Prieto-Díaz, 1993). According to Prieto-Díaz, programmers have been reusing parts of existing software in new software developments. This is however, done informally and very much ad-hoc (Sametinger, 1997; Prieto-Díaz, 1993). Prieto-Díaz asserts that, optimal pay-off from reuse can only be achieved, if reuse is conducted systematically and formally. Systematic reuse entails development *with* reuse and development *for* reuse (Sametinger, 1997). The former involves integrating existing components in new contexts, whilst the latter involves developing reusable components. According to Sametinger, development *for* reuse requires that we focus on the attributes that influence reusability, so that the developed components may have the desired levels of reusability. In addition, planned reuse requires organizations to establish reuse guidelines and programs; measure the reuse performance; and most importantly establish proper organizational structures.

3. Contribution of the Study

There are published claims of widespread ad-hoc reuse within the software Engineering industry (Sametinger, 1997; Prieto-Díaz, 1993). This result to a common conclusion that, the software Engineering industry is still grappling with software development challenges that reuse is intended to solve. The purpose of this paper is to report the findings of an empirical study that surveyed software developers who had knowledge in OO software development. The findings of the survey shed light on the nature of software reuse and reusability in industry, as well as the factors that impede successful reuse. This paper makes the following key contributions to the software Engineering community:

- It provides the state of affairs with regards to software reuse and reusability in the Kenyan software Engineering industry—forming a basis for further study on how to improve reuse and reusability.
- It provides information on the perceptions and awareness of software developers with regards to the role of software measurement in software quality.
- It identifies problems that are associated with the current reusability assessment methods. We give recommendations on how to improve reusability based on the identified problems.

4. Research Methodology

The aim of the research was to identify methods that Object-oriented (OO) software developers use in assessing the reusability of software components, as well as analyze their efficacy—that is, establishing the state of affairs in OO software reuse and reusability assessment. The survey research design was adopted for the study, since it is suitable for descriptive research. The study largely employed quantitative methods to collect data from respondents, using interview schedules—consisting of mostly closed ended questions. Qualitative methodology was also used to gain an in-depth understanding of other complex issues influencing OO reuse and reusability assessment, which would not have been understood, if only quantitative methodology was adopted. The target population for the study was made up of all OO software developers in the republic of Kenya. The respondents were selected from software development companies situated in Nairobi—the capital city of Kenya. Nairobi was selected as a study area for reasons of practicability, efficiency, and ease of access.

The researchers purposely targeted OO software developers because the study revolved around reuse and reusability assessment in OO software development, and the researchers believe that OO developers had sufficient knowledge on the subject matter, hence reliable for the study. Since the study focused on a population with largely similar characteristics, homogeneous sampling technique was used to draw a population sample from the target population—where a population sample of fifty-four (54) respondents was considered for the study. Homogeneous sampling technique is a type of purposive sampling that picks up a small sample with similar characteristics to describe some particular subgroup in depth (Kombo & Tromp, 2006). Data that was obtained from the respondents was presented in numbers and analyzed statistically in order to; classify, summarize, as well as draw meaningful conclusions and inferences. The Statistical Package for Social Sciences (SPSS) was the analytical tool used for data analysis.

5. Data Analysis and Discussions

The interview schedules that were used to collect data from respondents had the same structure and questions—in order to provide consistent results, as well as enable statistical comparisons of different cases. The schedules had five major sections, viz., Programmer's general background; Software development cycle; Organization's software reuse practice; Software development cost, Effort and productivity; Software reusability assessment; and, Software metrics.

5.1. Programmer's general background

This section captured three aspects: years the respondents had worked as programmers, programming languages known by the respondents and other software development related skills that the respondents had besides programming. Statistics about the former and the latter aspects were of interest, because the researchers believe that the practice of reuse and reusability assessment may be influenced by the experience of programmers, and other software development related skills—such as Software engineering (SE), Object-Oriented Analysis and Design (OOAD), System Analysis and Design (SAD), Software project management (SPM) etc, that the programmers possess.

5.1.1. Respondents' Experience in programming

From the data collected, 6 (11.1%) of the respondents had worked as programmers for 1 year and below. The respondents who had an experience of 2 – 5 five years as programmers were 28 (51.9%). 17 (31.5 %) of the respondents had an experience of between 6 – 10 years, and 3 (5.6%) of the respondents had an experience of between 11 – 15 years. These statistics are shown in table 1.

Table 1. Statistics on the experience of programmers in Years

Experience in years	**No. of Respondents**
1 and under	6 (11.1%)
2 - 5	28 (51.9%)
6 - 10	17 (31.5%)
11 - 15	3 (5.6%)
Total	**54 (100%)**

Source: Field Data, June 2014.

5.1.2 Respondents' Software development related skills

Statistics about other software development related skills possessed by the respondents are presented in table 2.

Table 2. Statistics on S/w Development Skills possessed by respondents besides Programming

Knowledge	No.	Percent (%)
OOAD, SAD	13	24.1
OOAD, SE	1	1.9
OOAD, SE, SAD	22	40.7
OOAD, SE, SAD, CASE	1	1.9
OOAD, SE, SAD, Mobile software Development	1	1.9
OOAD, SE, SAD, Project Management	1	1.9
OOAD, SE, SAD, Project Scheduling	1	1.9
OOAD, SE, SAD, SPM	1	1.9
OOAD, SAD	1	1.9
SAD	6	11.1
SAD, Database Programming	1	1.9
SAD, Project Management	1	1.9
SE	1	1.9
SE, SAD	3	5.6
Total	54	100.0

Source: Field Data, June 2014

A closer look at the results in table 2 reveals that, majority—32 (59.3%) of the respondents, had software engineering skills. Therefore we can conclude that, at least 59.3% of the respondents had some theoretical knowledge on software metrics. This is based on the fact that; software measurements and metrics are core areas in Software Engineering (Pressman, 2005).

5.2. Software Development Cycle

This section was intended to establish the state of affairs with regards to the activities of the Software development cycle, i.e. requirements modeling, design, coding, testing and maintenance of OO software. Most importantly, it explored issues that deal with technology use, reuse practice, design guidelines, and challenges experienced by respondents while undertaking the activities of the development cycle.

5.2.1. Technology Use in software development

With regards to technology use to support software design and development, 34 (63%) of the respondents indicated that they used computer-aided software engineering (CASE) tools in requirements modelling and analysis, 29 (53.7%) of the respondents indicated that they used computerized support in class design, whilst 30 (55.6%) of the respondents indicated that they used code generators to translate code into design. A summary of the data analysis results with this regards are displayed in table 3.

Table 3. Statistics on Technology Use in Software Development

Tool/Technology	No. of Respondents Using Technology	Total number of respondents
CASE tools in Requirement modeling	34 (63%)	54 (100%)
Computer support for Class design	29 (53.7%)	54 (100%)
Use of Code Generators	30 (55.6%)	54 (100%)

Source: Field Data, June 2014

According to (Mahapatra, Das, & Pradhan, 2012), CASE tools are often used in various stages during the systems development life cycle to improve software quality and productivity. This perspective is explored by creating contingency tables, where the levels of satisfaction with respect to software quality, productivity, and effort, are compared for respondents who use certain CASE tools and those who don't.

Table 4, shows the levels of satisfaction with regard to quality of software between respondents who use CASE tool in requirements modeling and those who don't.

Table 4. Satisfaction levels w.r.t s/w quality for those using CASE tools in requirements modeling & those who don't

		Satisfied with quality of developed s/w					**Total**
		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	
Use of CASE tools in requirement modeling and analysis	Yes	**0** (0%)	**1** (2.9%)	**7** (20.6%)	**13** (38.2%)	**13** (38.2%)	**34** (100%)
	No	**1** (5%)	**1** (5%)	**5** (25%)	**10** (50%)	**3** (15%)	**20** (100%)
Total		**1** (1.9%)	**2** (3.7%)	**12** (22.2%)	**23** (42.6%)	**16** (29.6%)	**54** (100%)

Source: Field Data, June 2014

By observing the marginal totals of the above contingency table, it can be seen that, the satisfaction levels (with respect to software quality) for respondents who use CASE tools in requirement modeling were higher than for those who don't use CASE tools.

For the respondents who use CASE tools, 13 (38.2%) *Agreed* that they were satisfied with the quality of developed software, whilst 13 (38.2%) *strongly agreed.* A small proportion, 1(2.9%) of the respondents *disagreed* with the assertion, whilst 7(20.6%) of the respondents were *neutral*. For respondents who don't use CASE tools, 10 (50%) *agreed*, 3 (15%) *strongly agreed*, 1(5%) *disagreed*, another 1 (5%) *strongly disagreed*, whilst 5 (25%) were *neutral*.

The same situation replays when the levels of satisfaction with respect to time and effort needed to test and modify s/w—for those using computerized support in class design are compared with levels of satisfaction of those of who don't. The cross tabulation analysis results are displayed in table 5. By observing the marginal totals of the contingency tables it is evident that the levels of satisfaction is higher for respondents who use computerized support in class design as compared to those respondents who don't.

Table 5. Satisfaction levels w.r.t time & effort needed to test & modify s/w for those who used computerized support in class design & those who don't

		Satisfied with time & effort required to test deliver & modify delivered s/w					Total
		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	
Use Computerized support in class design	Yes	**1** (3.4%)	**4** (13.8%)	**8** (27.6%)	**11** (37.9%)	**5** (17.2%)	**29** (100%)
	No	**2** (8%)	**10** (40%)	**4** (16%)	**3** (12%)	**6** (24%)	**25** (100%)
Total		**3** (5.6%)	**14** (25.9%)	**12** (22.2%)	**14** (25.9%)	**11** (20.4%)	**54** (100%)

Source: Field Data, June 2014

In the case of respondents who use computerized support in class design, 11 (37.9%) of respondents *agreed* that they were satisfied with time and effort required to deliver, test, and modify s/w, 5 (17.2%) of the respondents *strongly agreed*. Those who *disagreed* and *strongly disagreed* were 4 (13.8%) and 1 (3.4%), respectively. On the other hand, the satisfaction levels for respondents who don't use computerized support in class design were surpassed by their dissatisfaction levels. A relatively low proportion 3 (12%) *agreed* that, they were satisfied with time and effort required to deliver, test, and modify s/w, while 6 (24%) *strongly agreed*. A significantly high proportion; 10 (40%) of the respondents *disagreed* with the assertion, 2 (8%) of the respondents strongly *disagreed*, whilst 4 (16%) were neutral.

However, the situation is quite different when it comes to s/w quality and use of code generators. The levels of satisfaction with respect to s/w quality were not significantly different for respondents who use code generators and those who don't. The total number of respondents who *agreed* and those who *strongly agreed* that they were satisfied with quality of developed software were 22 (73.3%)—for those who use code generators, and 17 (70.8%)—for those who don't. On the other hand, the total number of respondents who *disagreed* and those who *strongly disagreed* were 2 (6.7%)—for those who use code generators, whilst 1 (4.2%) *disagreed*. Respondents who were neutral were, 6 (20%), and 6 (25%) for those who use code generators and those who don't respectively. These statistics are shown in table 6.

Table 6. Satisfaction levels w.r.t s/w quality for those using code generators & those who don't

		Satisfied with quality of developed s/w					Total
		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	
Use of Code generators	Yes	**1** (3.3%)	**1** (3.3%)	**6** (20%)	**16** (53.3%)	**6** (20.0%)	**30** (100%)
	No	**0** (0.0%)	**1** (4.2%)	**6** (25%)	**7** (29.2%)	**10** (41.7%)	**24** (100%)
Total		**1** (1.9%)	**2** (3.7%)	**12** (22.2%)	**23** (42.6%)	**16** (29.6%)	**54** (100%)

Source: Field Data, June 2014

One important conclusion that can be drawn from table 6 is that, software quality is not largely dependent on code quality: other factors such as quality of design, and how well the software meets requirements, are very important factors when it comes to software quality.

5.2.2. Reuse practice within the development cycle

The reuse practice within the software development cycle was also explored—by asking respondents whether they reused requirements documents, design, and code when developing new software. The summary of the data analysis results with this regards is given in the table 7.

Table 7. Statistics on Components Reuse

Component Reused	**No. of Respondents Reusing the Component**	**Total Number of Respondents**
Requirements Documents	41 (75.9%)	54 (100%)
Design	40 (74.1%)	54 (100%)
Code	54 (100%)	54 (100%)

Source: Field Data, June 2014

As it can be seen from the above table, code reuse has the most interesting statistics, where 54 (100%) of respondents indicated that they often reused code from existing software. It was in the interest of the researchers to know the most significant challenges that respondents faced when reusing code. The challenges cited by the respondents mostly revolved around issues that deal with: code understandability; integration of existing code into the new system code; debugging errors associated with the reused code; difficulty in finding code that perfectly fits into the new system code; insufficient in-text documentation (comments).

5.2.3. Challenges experienced in s/w testing and maintenance vs. s/w design guidelines

The researchers also sought to know whether or not the respondents faced any challenges with regards to software testing and maintenance. Majority of respondents, 36 (66.7%) indicated that they faced challenges when testing and maintaining software. This is shown in table 8.

Table 8. Statistics on s/w testability and maintainability challenges

Statement	**Response**	**Number of respondents**
Experience challenge when testing and maintaining s/w?	Yes	36 (66.7%)
	No	18 (33.3%)
	Total	54 (100%)

Source: Field Data, June 2014

The most prominent testability and maintainability challenges cited by the respondents include: *difficulty in modifying existing components, time constraints, difficulty in debugging, difficulty in testing and maintaining software developed elsewhere, generating sufficient test cases and test data, lack of experience in testing, poor documentation, lack of testing tools, lack of a clear testing criteria, and integration testing*.

One possible cause for some of the challenges experienced during software testing and maintenance is complexity of some of the developed software (i.e. developed software having little understandability). This explanation is consistent with the assertion by (Laird & Brennan, 2006) that, unnecessary complexity brings about problems such as additional defects, difficulty in understanding code, difficulty in debugging, and maintainability issues.

According to (Ghezzi, Jazayeri, & Mandrioli, 2003), some guidelines can be followed in order to produce less complex software. To explore this aspect, respondents were asked to indicate whether they followed some OO design criteria/guidelines: coupling, cohesion, and inheritance; when designing software. Table 9 shows the statistics of the responses that were gathered from the collected data.

Table 9.Statistics on the OO design guidelines/criteria followed

OO design guideline/criteria	**No. of respondents who follow the guideline/criteria**	**No. of respondents who don't follow the guideline/criteria**	**Total No. of Respondents**
Cohesion and coupling criteria	32 (59.3%)	22 (40.7%)	54 (100%)
Control inheritance hierarchy	41 (75.9%)	13 (24.1%)	54 (100%)

Source: Field Data, June 2014

When the statistics on tables 8 and 9 are compared, it can be observed that some of the respondents who follow design guidelines—that are intended to produce understandable software that are easy to test and maintain, also experience significant challenges when testing and maintaining software. The true picture of this phenomenon is revealed by creating contingency tables.

Table 10. Cohesion and coupling criteria in class design vs. s/w Testability and maintainability challenges

		Experience Challenge When testing and maintaining s/w		Total
		Yes	No	
Follow Cohesion and Coupling Criteria in Class design	Yes	20 (62.5%)	12 (37.5%)	32 (100%)
	No	16 (72.7%)	6 (27.3%)	22 (100%)

Source: Field Data, June 2014

From table 10, majority 20 (62.5%) of the respondents who follow cohesion and coupling criteria in class design indicated that they experience significant challenges when testing and maintaining software; however, this percentage is higher by 10.2%, for respondents who don't follow cohesion and coupling criteria. This situation also holds when it comes to control of inheritance hierarchy during class design. As it can be seen from table 11, the percentage of respondents who experience significant challenge when testing and maintaining software is lesser for respondents who control inheritance hierarchies during class design, as compared to that of respondents who don't.

Table 11. Control of Inheritance hierarchy during class design vs. S/w testing and maintenance challenges

		Experience Challenge When testing and maintaining s/w		Total
		Yes	No	
Control inheritance hierarchy during class design	Yes	24 (58.5%)	17 (41.5%)	41
	No	12 (92.3%)	1 (7.7%)	13

Source: Field Data, June 2014

By observing the above contingency tables, we can conclude that, challenges that are related to software testing and maintenance could be resolved if developers follow design guidelines such as, following cohesion and coupling criteria; and, controlling of inheritance hierarchies. However, the marginal totals in table 11 show that a significantly high number of respondents who follow design guidelines face testability and maintainability challenges. This is an indication that the guidelines are not followed to the latter—or rather an objective way of assessing how well these guidelines are followed is lacking.

To further investigate the effect of following design guidelines on software testing and maintenance, linear correlation analysis between following design guidelines; and software testability and maintainability challenges is conducted. The linear correlation analysis results in table 12 show that, there is a negative correlation (of –

0.306 with a ***p*** value of .024) between *control of inheritance hierarchy during class design*, and the *Experience of Challenges when testing and maintaining s/w*.

Table 12. Correlation between following OO design criteria and S/w testing challenges

		Experience Challenge When testing and maintaining s/w	Follow Cohesion and Coupling Criteria in Class design	Control inheritance hierarchy during class design
Experience Challenge When testing and maintaining s/w	Pearson Correlation	1	-.107	-.306*
	Sig. (2-tailed)		.443	.024
	N	54	54	54
Follow Cohesion and Coupling Criteria in Class design	Pearson Correlation	-.107	1	.415**
	Sig. (2-tailed)	.443		.002
	N	54	54	54
Control inheritance hierarchy during class design	Pearson Correlation	-.306*	.415**	1
	Sig. (2-tailed)	.024	.002	
	N	54	54	54

Source: Field Data, June 2014

During data collection, the respondents replied in the affirmative or otherwise, whether they control inheritance hierarchies during class design, and if they experienced challenges during software testing and maintenance. During the coding of variables, 1 represented Yes, whilst 2 represented No—for all cases. Therefore, the negative correlation between the variables is an indication that, when the average value for one variable tends to 1 (Yes), the average value for the other will tend to 2 (No), i.e. the more the number of respondents who control inheritance hierarchies during class design, the lesser the respondents who face testability and maintainability challenges.

The fact that table 12 shows that there is no correlation between following of cohesion and coupling criteria in class design; and, testing and maintenance challenges, does not mean that there is no relationship between the two. To study their relationship, partial correlation between *control of inheritance hierarchy during class design*, and *challenges experienced when testing and maintaining s/w* is studied, where the effect of following cohesion and coupling criteria in class design is controlled on the two variables. The partial correlation results are shown in table 13.

Table 13. Partial correlation between control of inheritance hierarchy and S/w testing challenges

Control Variables			Control inheritance hierarchy during class design	Experience Challenge When testing and maintaining s/w
Follow Cohesion and Coupling Criteria in Class design	Control inheritance hierarchy during class design	Correlation	1.000	-.290
		Significance (2-tailed)	.	.035
		df	0	51
	Experience Challenge When testing and maintaining s/w	Correlation	-.290	1.000
		Significance (2-tailed)	.035	.
		df	51	0

Source: Field Data, June 2014

As it can be observed from the above table, the partial correlation is somewhat smaller than the simple correlation. This suggests that *following cohesion and coupling criteria in class design* partly contributed to the simple correlation between control of inheritance hierarchy during class design, and the experienced testability and maintainability challenges. This means that following cohesion and coupling criteria eliminates some of the challenges associated with software testing and maintenance.

5.3. Organization's Software Reuse Practice

This section sought to explore the efficacy of software reuse policies and the software Reuse practice within the organizations. From the data collected, only 13 (24.1%) of the respondents indicated that their organizations had a software reuse policy in place. On the other hand, 54 (100%) of the respondents indicated that they reused parts of existing software in new software developments. These statistics are shown in table 14.

Table 14. Statistics on the Organizations Software reuse policy and practice

Aspect inquired about	**No. of responses in the affirmative**	**Total No. of respondents**
Reuse policy in place within the organization	13 (24.1%)	54
Reuse parts of existing s/w in new s/w development	54 (100%)	54

Source: Field Data, June 2014

To examine the efficacy of the software reuse practice within the organizations, respondents were to indicate the extent to which they agreed or disagreed to some statements that are related to software reuse practice within their organizations. The respondents were given five options to choose one: *Strongly Disagree (1), Disagree (2), Neutral (3), Agree (4), strongly Agree (5).*

Table 15. Respondents' Perceptions on the Efficacy of software reuse policies and reuse practice

Statement	**N.**	**Strongly Disagree**	**Disagree**	**Neutral**	**Agree**	**Strongly Agree**
Cases of developing software from scratch have significantly diminished over time	54	4 (7.4%)	6 (11.1%)	2 (3.7%)	21 (38.9%)	21 (38.9%)
The time and effort required to modify available classes within the organization to fit new reuse contexts is often insignificant as compared to creating new classes	54	1 (1.9%)	3 (5.6%)	10(18.5%)	24(44.4%)	16(29.6%)
The cost and effort for developing software has significantly diminished over time.	54	2 (3.7%)	7 (13.0%)	15(27.8%)	13(24.1%)	17(31.5%)
I prefer developing classes from scratch than reuse classes that are developed by my colleagues	54	5(9.3%)	3(5.6%)	18(33.3%)	16(29.6%)	12(22.2%)

Source: Field Data, June 2014

From table 15, the total number of respondents who *agreed* and those who *strongly agreed* with the first three statements were 42 (77.8%), 40 (74%) and 30 (55.6%) respectively. However, the fourth statement: *I prefer developing classes from scratch than reuse classes that are developed by my colleagues*, got interesting responses. Most of the respondents (51.8%) prefer developing classes from scratch than reuse classes that are developed by others. The total percentage of respondents who *disagreed* and those who *strongly agreed* is only 14.9%. This may be as a result of two factors: (i) the not-invented-here syndrome—a situation where developers feel hindered in their creativity and independence if they reused someone else's software (Sametinger, 1997), and (ii) some of the existing components are difficult to reuse; due reusability related issues.

5.4. Software Development Cost, Effort and Productivity

This section explored the payoff from reuse within the organizations. Four statements that are related to software development cost, effort and productivity were put forward, and respondents indicated the extent to which they agreed or disagreed with the statements.

Table 16. Respondents' satisfaction levels on the Payoff of software reuse practice

Statement	N.	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I am satisfied with the time and effort that is always required to, test, deliver and maintain new software to our clients.	54	3(5.6%)	14(25.9%)	12(22.2%)	14(25.9%)	11(20.4%)
I am satisfied with budget and cost aspects for developing new software applications and their maintenance.	54	8(14.8%)	10(18.5%)	21(38.9%)	12(22.2%)	3(5.6%)
I am satisfied with the quality of new software applications we develop as an organization.	54	1(1.9%)	2(3.7%)	12(22.2%)	23(42.6%)	16(29.6%)
I am satisfied with the overall productivity of developers in the organization.	54	1(1.9%)	9(16.7%)	16(29.6%)	15(27.8%)	13(24.1%)

Source: Field Data, June 2014

Table 16, shows that the total number of respondents who *agreed* and those who *strongly agreed* with the first and the second statements were less than 50%, i.e. that is 46.3% and 27.8% respectively. This is in spite of the fact that all 54 (100%) of the respondents indicated that they reused parts of existing software when developing new software. This is an indication that, organizations are not gaining maximum payoff from reuse. It is also reflection of the informal nature of the reuse practice across the organizations. This can be seen from table 14, where only 24.1% of respondents indicated that their organizations had a reuse program/policy in place. According to (Prieto-Díaz, 1993), substantial pay-off from reuse is only achieved if conducted systematically and formally.

Although majority 39 (72.2%) of the respondents indicated that they were satisfied with the quality of new software applications that they developed, majority were not satisfied with the effort and time it took to deliver software. This means that it took a lot of effort and time to achieve the desired quality.

5.5. Software Reusability Assessment

This section explored reusability assessment within the organizations, and respondents' understanding of the concept of reusability.

5.5.1. Reusability Assessment within the organizations

Respondents were required to respond in the affirmative or otherwise, whether they ascertained reusability when developing *for* reuse and *with* reuse. The results of the responses are shown in table 17, where majority 33 (61.1%) of the respondents indicated that they did not ascertain reusability of components when developing *for* or *with* reuse.

Table 17. Statistics on Reusability assessment

Statement	N.	Yes	No
Do you always ascertain if classes are reusable when developing or reusing them?	54	21 (38.9%)	33 (61.1%)

Source: Field Data, June 2014

Further, respondents who indicated that they ascertained reusability when developing *for* reuse and *with* reuse were asked to give major characteristics that classes must have for them to be reusable. Some of the most prominent cited characteristics include: *Ability of a component to perform required functionality, Ease of testing, Portability across different platforms, Proper use of abstraction and inheritance, Easy to understand and adapt, Class Independence, Proper documentation, Have public accessor methods and be part of a hierarchy with an abstract/interface, Consistency in naming methods, Should be as generic as possible, Well commented and documented, Tested and used before (Reuse history).*

5.5.2. Respondents' understanding on OO Reusability

In order to establish the level of understanding of the respondents with regards to OO reusability, those respondents who stated that they ascertained reusability when developing *for* or *with* reuse were required to state whether they were aware of how OO design features were related to the reusability attributes that they stated. The cross-tabulation analysis results are shown in table 18.

Table 18. Respondent's awareness of OO design features and reusability characteristics

		Aware of how OO features influence the listed reusability characteristics		Total
		Yes	No	
Ascertain if classes are reusable when developing for or with reuse	Yes	18 (85.7%)	3 (14.3%)	21

Source: Field Data, June 2014

From the above contingency table, majority; 18 (85.7%) of respondents who indicated that they ascertained reusability were aware how OO design features were related with reusability attributes, while only 3 (4.3%) of the respondents did not know.

5.5.3. Reusability assessment methods used by Respondents

The methodology used by respondents to ascertain reusability was also explored, where respondents were asked to state whether they had formal methods for ensuring that classes they develop possessed reusability characteristics. The output of the cross tabulation analysis is shown in table 16, where it was observed that only 4 (19%) of the respondents had formal methods for ensuring that components possess reusability characteristics, whilst majority 17 (81%), of the respondents had no formal methods.

Table 19. Existence of reusability assessment methodology

		Have formal methods for ascertaining if classes have reusability characteristics		Total
		Yes	No	
Ascertain if classes are reusable when developing for or with reuse	Yes	4 (19%)	17 (81%)	21

Source: Field Data, June 2014

Respondents who stated that they had formal methods for assessing reusability also cited methods that they used. The cited methods were largely subjective and they include: *Observing/checking source code, reading documentation, Intuition,* and *checking inline documentation (comments).*

5.6. Software Metrics

This section explored reusability measurement within the software development industry. It focused on organizations' software measurement policy, respondent's experience with software metrics, and reusability measurement. From the data collected, a small proportion 12 (22.2%) of the respondents indicated that organizations they worked for had a software measurement program/policy, whereas 42 (77.8%) of the respondents indicated that their organizations had no software measurement program/policy in place. Statistics with this regards are shown in table 20.

Table 20. Software measurement policy/program within organizations

Statement	**Response**	**No. of respondents**
Does your organization have a software measurement program/policy?	Yes	12 (22.2%)
	No	42 (77.8%)
	Total	54 (100%)

Source: Field Data, June 2014

With regards to respondents' experience with software metrics, respondents were asked to indicate their experience with metrics in their occupation. Five options were given where respondents were required to choose one. A summary of the gathered responses is given in table 21.

Table 21. Respondent's experience with metrics

Responses	No. of Respondents
Never heard about them	6 (11.1%)
Heard about them but never used them	11 (20.4%)
I have knowledge on Metrics but never used them	24 (44.4%)
I have used Metrics before but stopped using them	1 (1.9%)
I always use software Metrics	12 (22.2%)
Total	54 (100%)

Source: Field Data, June 2014

Although 12 (22.2%) of the respondents indicated that they always used software metrics, none of them respondent in the affirmative when asked whether or not they measured reusability when developing *for* or *with* reuse, as shown in table 22. Further explanation was sought and it was established that those who use metrics, used software parametric models and parametric estimation tools to estimate projects' duration, effort required in developing software, and cost of developing software.

Table 22. Statistics on Reusability Measurement

Statement	Response	Number of Respondents.	Total Number of Respondents
Do you measure the reusability of classes when developing for or with reuse?	No	54 (100%)	54 (100%)

Source: Field Data, June 2014

Further it was in the interest of the researchers to establish why respondents don't measure reusability. Some of the most interesting responses that were given include: *As a programmer, the only concern is to developing working software delivered within time; Do not know how to measure reusability; Don't know how it's helpful; Have limited knowledge; Have never considered measuring software to be useful; Insufficient knowledge; It is hard to apply metrics in practice; Lack of Practical knowledge on how to measure reusability; Lack of reusability measurement tools; Measurement policy does not cover product quality; Tight deadlines hence focus is on delivering software on time by all means…*

The cited reasons can be summarized as: Lack of sufficient knowledge on software metrics and software quality measurement, Organizations' measurement policies failed to cover some quality aspects such as reusability, Lack of parametric tools for measuring reusability, and time constraints.

6. Summary of the key findings and Conclusions

The purpose of the survey was to establish the state of affairs within the software development industry, regarding OO software reuse and reusability assessment. The benefits of reuse as well the impediments to successful reuse were also explored.

6.1. Experience and skills of respondents

From the data collected, it was established that, 6 (11.1%) of the respondents had worked as programmers for 1 year and below, 28 (51.9%) had an experience of between 2 – 5 years, 17 (31.5 %) had an experience of between 6 – 10 years, while 3 (5.6%) of the respondents had an experience of between 11 – 15 years. It was also established that, apart from programming, at least 32 (59.3%) of the respondents had Software Engineering skills. Therefore, it can be concluded that at least the 32 (59.3%) had a theoretical background on software measurement and metrics. This is based on the fact that; software measurements and metrics are core areas in Software Engineering.

With regards to respondents' experience with software metrics, the survey revealed that 6 (11.1%) of respondents have never heard about metrics, 11 (20.4%) indicated that they have heard about metrics but they have never used them, 24 (44.4%) indicated that had knowledge on Metrics but they have never used them, 1 (1.9%) indicated that they stopped using metrics, and finally 12 (22.2%) of the respondents indicated that they always used metrics.

6.2. Technology Use in Software Development

It is evident from the survey that majority of developers use technology (CASE tools) to support the process of software developments. Some benefits and limitations of technology were also identified. The survey showed that the process of software development can be improved to some extent—if technology is incorporated. Analysis of the collected data showed that the satisfaction levels w.r.t software quality, as well as time and effort needed to test and modify s/w; were higher for respondents who use technology support in requirement modeling, and in class design as compared to those who don't use technology. However, the levels of satisfaction with respect to s/w quality were not significantly different for respondents who use code generators and those who don't. This means that, software quality is not dependent on code quality alone; therefore, other factors such as quality of design, and how well the software meets requirements, are very important and must be focused on when it comes to software quality.

6.3. Challenges in code reuse, software testing and maintenance

Most of the respondents who reused code indicated that they faced significant challenges when reusing code. Some of the cited challenges include: difficulty in understanding code, integration of existing code into the new system code, and debugging errors associated with the reused code. Also, majority 36 (66.7%) of the respondents indicated that they faced challenges when testing and maintaining software. The most prominent challenges that were cited include: difficulty in modifying existing components, time constraints, difficulty in debugging, difficulty in testing and maintaining software developed elsewhere, generating sufficient test cases and test data, lack of experience in testing, poor documentation, lack of testing tools, lack of a clear testing criteria, and integration testing.

One possible cause of some of the challenges experienced during code reuse, software testing, and maintenance is complexity of the developed software—which can be resolved by following design guidelines that can lead to producing more understandable software. Analysis of the survey data showed that the proportion of respondents who faced challenges when testing and maintaining software was lower for respondents who follow design guidelines, as compared to those who don't follow these guidelines. However, further analysis of the survey data revealed that quite a high number of respondents who follow the design guidelines still face software testing and maintenance related challenges—raising the question of how well these guidelines are followed. Therefore, it is a necessary condition for developers to follow design guidelines but it is not sufficient in ensuring that resulting software are of desired quality. Thus, in addition to developers following design guidelines, they must have an objective way of assessing how well the guidelines are followed. One of such ways is by using appropriate software metrics.

6.4. Impediments to successful Reuse in the Organizations

With regards to component reuse, the survey revealed that, 41 (75.9%) of the respondents reused Requirements Documents, 40 (74.1%) reused design, whilst 54 (100%) of the respondents reused Code. However, only 13 (24.1%) of the respondents indicated that their organizations had a software reuse program/policy in place. This means that, the reuse practice is largely being conducted in an opportunistic manner. This is one of the major impediments to successful reuse, and it is reflected in the low satisfaction levels of the respondents with respect to the time, effort, and cost aspects of developing, testing, and maintaining of software.

Although majority of respondents indicated that they enjoyed some benefits from reuse—such as; reduction of cost, time, and effort required to develop, test, and modify software, a large proportion (51.8%) of the respondents indicated that they preferred developing classes from scratch than reuse classes that are developed by others. This may be as a result of two factors: (i) the not-invented-here syndrome and (ii) some of the existing components are difficult to reuse. The former is a situation where developers feel hindered in their creativity and independence if they reused someone else's software. If this problem is not dealt with, it can hinder successful reuse; thus, organizations need to deal with it—in order to realize maximum benefits from reuse. The not-invented-here-syndrome can be dealt by: (i) developers changing their perceptions with regards to reuse, and (ii) organizations devising ways of encouraging developers to reuse. The difficulty of reusing existing components on the other hand, is due to reusability related issues, i.e. some of the existing components have little or no reusability. This issue can be dealt with by organizations adopting planned/systematic reuse; which involves developing *for* reuse, and development *with* reuse. The former involves developing components with reuse in mind—where reusability of the component is often the main focus, whilst the latter involves identifying and reusing existing components.

6.5. Software Reusability and Reusability Assessment

The reusability aspect of developed software components was also explored, in order to understand some arising issues that are related to reuse within the organizations. From the data collected, it was observed that majority 33 (61.1%) of the respondents did not assess reusability of components when developing *for* or *with* reuse. We can then conclude that some of the components that are developed are often hard to reuse (i.e. they possess less or no reusability), because; if reusability of components is not assessed, then it could not be possible to know whether or not he components possess the required degree of reusability. This is manifested in the low satisfaction levels of the respondents with respect to time, effort, and cost aspects; of developing, testing, and maintaining software—issues that effective reuse should solve.

6.6. Challenges in Reusability Measurement

One possible reason for the low number, 21 (38.9%) of respondents who ascertain reusability when developing software *for* and *with* reuse is that, majority of the organizations had no software measurement programs/policies in place, while the existing measurement programs/policies did not cover the reusability aspect. From the data collected, it was observed that majority, 42 (77.8%) of the respondents indicated that their organizations don't have software measurement policies and programs. Although 12 (22.2%), of the respondents indicated that their organizations had software measurement policies and programs in place, the said policies don't cover reusability assessment. Developers in these organizations indicated that they mainly used parametric models and parametric estimation tools to estimate projects' duration, effort and cost of developing software.

Another challenge that developers face in reusability assessment is lack of objective methods of measuring reusability. From the data collected, it was observed that only 4 (19%) of the respondents who indicated that they ascertained reusability when developing for and with reuse, had a way of ensuring that components possessed reusability characteristics, whilst majority17 (81%) of the respondents had no formal methods. The reusability assessment methods cited by the respondents were largely subjective and they included: Observing/checking source code, reading documentation, Intuition, and checking inline documentation (comments).

On the other hand, the reasons cited by respondents as to why they don't measure reusability can be summarized as: Lack of sufficient knowledge on software metrics and software quality measurement, Organizations' measurement policies failed to cover some quality aspects such as reusability, Lack of parametric tools for measuring reusability, and time constraints.

6.7. Recommendations

Software reuse has a high return on investment, if it is carried out in a planned and systematic manner—which requires that software being developed or being reused possess adequate reusability. Therefore, developers need to adopt objective ways of assessing reusability of components when developing for reuse, or when identifying components for reuse. The most rational way of achieving this is by using metrics. Metrics will provide indication of the status of reusability, thus providing a basis for improvement—in the case of inadequate reusability. Organizations are therefore required to establish comprehensive software measurement programs and policies that cover the aspect of reusability, whilst those organizations with measurement programs in place need to extend those programs to cover the reusability aspect. In addition, developers should, adopt metrics-based reusability assessment frameworks such as the ones proposed in, (Nyasente et al., 2014; Caldiera & Basili, 1991; Hristov et al., 2012; Washizaki et al., 2003). This will greatly improve component reusability, and reuse.

Lastly, the not-invented here syndrome can be dealt with—by organizations sensitizing their developers on the importance of reuse, as well as fostering the culture of reuse by rewarding developers who reuse.

References

AL-Badareen, A. B., Selamat, H. M., Jabar, M. A., Din, J., & Turaev, S. (2010). Reusable Software Components Framework. *Advances in Communications, Computers, Systems, Circuits and Devices*, (pp. 126-130). Puerto De La Cruz, Tenerife.

Caldiera, G., & Basili, V. R. (1991). Identifying and Qualifying Reusable Software Components. *IEEE Computer , 24*, 61-70.

Chawla, S., & Nath, R. (2013). Evaluating Inheritance and Coupling Metrics. *International Journal of Engineering Trends and Technology (IJETT) , 4* (7), 2903-2908.

Chidamber, S. R., & Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering , 20* (6), 476-493.

Dubey, S. K., & Rana, A. (2010). A Comprehensive Assessment of Object-Oriented Software Systems Using Metrics Approach. *(IJCSE) International Journal on Computer Science and Engineering , 2* (8), 2726-2730.

Farooq, S. U., Quadri, S. M., & Ahmad, N. (2011). SOFTWARE MEASUREMENTS AND METRICS: ROLE IN EFFECTIVE SOFTWARE TESTING. *International Journal of Engineering Science and Technology (IJEST) , 3* (1), 671-680.

Frakes, W. B., & Kang, K. (2005). Software Reuse Research: Status and Future. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING , 31* (7), 529-536.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2003). *Fundamentals of Software Engineering* (2nd Edition ed.). New Jersey: Prentice-Hall.

Gill, N. S., & Sikka, S. (2011). Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD. *Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD , 3* (6), 2300-2309.

Hristov, D., Hummel, O., Huq, M., & Janjic, W. (2012). Structuring Software Reusability Metrics. *International Conference on Software Engineering Advances.* IARIA.

Ilyas, M., & Abbas, M. (2013). Role of Formalism in Software Reusability's Effectiveness. *International Journal of Database Theory and Application , 6* (4), 119-130.

Kombo, D. K., & Tromp, D. L. (2006). *Proposal and Thesis Writing: An introduction.* Nairobi: Paulines Publications Africa.

Laird, L. M., & Brennan, M. C. (2006). *Software Measurement and Estimation A Practical Approach.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Mahapatra, D. K., Das, T. K., & Pradhan, G. (2012). An Integration of JSD, GSS and CASE Tools towards the Improvement of Software Quality. *International Journal of Engineering and Advanced Technology (IJEAT) , 2* (2), 306-312.

Nyasente, S. O., Mwangi, W., & Kimani, S. (2014). A Metrics-based Framework for Measuring the Reusability of Object-Oriented Software Components. *Journal of Information Engineering and Applications , 4* (4), 71-84.

Pressman, R. S. (2005). *Software Engineering: A practitioner's Approach* (6th ed.). McGraw-Hill.

Prieto-Díaz, R. (1993). Software Reuse: Issues and Experiences. *American Programmer , 6* (8), 10-18.

Sametinger, J. (1997). *Software Engineering with Reusable Components.* Berlin: Springer-Verlag.

Sommerville, I. (2011). *Software Engineering* (9th Edition ed.). Boston: Pearson Education.

Washizaki, H., Yamamoto, H., & Fukazawa, Y. (2003). A Metrics Suite for Measuring Reusability of Software Components. *International Software Metrics Symposium*, (pp. 211-223).