Evolution of PHP Applications: A Systematic Literature Review

https://doi.org/10.3991/ijes.v5i1.6437

Douglas Kunda
Mulungushi University, Kabwe, Zambia
dkunda@mu.ac.zm

Alinaswe Siame,
Mulungushi University, Kabwe, Zambia
alinaswe7@gmail.com

Abstract—This paper reviews, some of the research work done in the evolution of PHP applications that have been around and are vastly used. PHP is currently one of the most popular programming languages, widely used in both the open source community and in industry to build large web-focused applications and application frameworks. This review looks at how PHP applications have evolved in terms of the use of libraries, the software maturity, adoption of object-orientation paradigm, the evolution of complexity and security. The results suggest that these systems undergo systematic maintenance and evolution is helping the underlying programming language to grow.

Keywords—software evolution; web applications; object-orientated programing; software libraries; PHP; scripting language

1 Introduction

PHP is currently one of the most popular programming languages, widely used in both the open source community and in industry to build large web-focused applications and application frameworks. [1]

Eshkevar, Dos Santos, Cordy, & Antoniol also add that PHP is by far the most popular WEB scripting language, accounting for more than 80% of existing websites. However, Scripting languages such as PHP have been criticized as inadequate for supporting maintenance of largescale software projects. [2]

Kyriakakis & Chatzigeorgiou attempt to provide insight into the way that PHP applications evolved over time. They examined several aspects of their history including the amount of unused code, the removal of functions, the use of libraries, the stability of their interfaces, the migration to object-orientation and the evolution of complexity. This evolution is brought about because a web application (Build in PHP) evolve, new versions of programs, interactions and functionalities are added and existing ones are removed or modified. Web applications require configuration and programming attention to assure security, confidentiality, and trustiness of the published infor-

mation. During evolution of Web software, from one version to the next one, security flaws may be introduced, corrected, or ignored. [3] This paper also takes into consideration that PHP is dynamically typed, which means that variables take on the type of the objects that they are assigned, and may change type as execution proceeds. While some type changes are likely not harmful, others involving function calls and global variables may be more difficult to understand and the source of many bugs. Hack, a new PHP variant endorsed by Facebook, attempts to address this problem by adding static typing to PHP variables, which limits them to a single consistent type throughout execution. [4]

2 Software Evolution

Software evolution deals with the process by which programs are modified and adapted to their changing environment. The aim of Lehman's research was to formulate a scientific theory of software evolution. As any sound theory, it was meant to be based on empirical results and aimed at finding invariant properties to be observed on entire classes of software development projects. [5]

Software evolution is also a crucial ingredient of so-called agile software development processes, of which extreme programming (XP) is probably the most famous proponent. In brief, agile software development is a lightweight iterative and incremental (evolutionary) approach to software development that is performed in a highly collaborative manner and explicitly accommodates the changing needs of its stakeholders, even late in the development cycle, because this offers a considerable competitive advantage for the customer. In many ways, agile methods constitute a return to iterative and incremental development as practiced early in the history of software development, before the widespread use of the waterfall model. [6]

Mens & Demeyer go on to say, today software evolution has become a very active and well-respected field of research in software engineering, and the terms software evolution and software maintenance are often used as synonyms. For example, the international ISO/IEC 14764 standard for software maintenance, acknowledges the importance of pre-delivery aspects of maintenance such as planning.

2.1 Difficulties in Software Evolution

According to Mens & Demeyer the main difficulties of software evolution is that all artefacts produced and used during the entire software life-cycle are subject to changes, ranging from early requirements over analysis and design documents, to source code and executable code. This fact automatically spawns many sub disciplines in the research domain of software evolution, some of which are listed below:

Requirements evolution. The main objectives of requirements engineering are defining the purpose of a software system that needs to be implemented. Requirements evolve because requirements engineers and users cannot predict all possible uses of a system, because not all needs and (often mutually conflicting) goals of the various

stakeholders can be taken into account, and because the environment in which the software is deployed frequently changes as well.

Architecture evolution. Based on an (initial) description of the software requirements, the overall software architecture (or high-level design) and the corresponding (low-level) technical design of the system can be specified. These are inevitably subject to evolution as well.

Data evolution. In information systems and other data-intensive software systems it is essential to have a clear and precise description of the database schema.

Runtime evolution. Many commercial software systems that are deployed by large companies need to be constantly available. Halting the software system to make changes cannot be afforded. Therefore, techniques are needed to change the software while it keeps on running. This very challenging problem is known under a variety of terms, including runtime evolution, runtime reconfiguration, dynamic adaptation and dynamic upgrading.

Service-oriented architectures (SOA) provide a new paradigm in which a user oriented approach to software is taken. The software is developed in terms of which services are needed by particular users, and these users should be able to easily add, remove or adapt services to their needs. While this approach has many similarities with the component-oriented approach, services are only bound together at runtime, whereas components are statically (i.e., at design time) composed together. A service-oriented approach thus promises to be inherently more flexible than what is available today. This is crucial, especially in e-commerce applications, where rapid and frequent change is a necessity in order to respond to, and survive in, a highly competitive market.

Language evolution. When looking at languages (whether it be programming, modelling of formal specification languages), a number of research directions come to mind. The first one is the issue of co-evolution between software and the language that is used to represent it. Both are subject to evolution, albeit at different speed. The second challenge is to provide more and better support for evolution in the context of multi-language software systems. A third challenge is to improve the design of languages to make them more robust to evolution (e.g., traits). This challenge has always been the main driver of research in design of new computer languages. Unfortunately, every new programming paradigm promises to improve the software development process but introduces its own maintenance problems. This was the case for object-oriented programming (where the inheritance hierarchy needs to be mastered and kept under control when evolving software), aspect-oriented programming (where aspects need to be evolved next to the base code, component oriented programming, and so on. In general, every new language or technology should always be evaluated in the light of its potential impact on the software's ability to evolve [6].

2.2 PHP Applications

The major reason in selecting acknowledged projects with a long history, large number of committers and even larger number of users. According to Samoladas, Angelis, & Stamelos the majority of open-source projects are abandoned after a short

time period, rendering them inappropriate for systematic analysis of programming and maintenance habits.

The case study has been conducted by Kyriakakis & Chatzigeorgiou on the following five open source projects implemented in PHP:

Wordpress: The most popular blogging software; it has a vast community of both contributors and active users.

Drupal. One of the most advanced CMS (Content Management System). It is also characterized by a large and active community.

PhpBB: One of the most widely used forum software.

MantisBt: Probably the most popular bug tracking application written in PHP.

PhpMyAdmin: The well-known MySQL administration tool.

In Figure 1 Kyriakakis & Chatzigeorgiou show some statistics about the selected projects. Cumulatively, we have studied 390 official releases aggregating to 50 years of software evolution.

Project	Years	First		Last		Number of Releases
		Release	Year	Release	Year	
WordPress	9	1.5	2005	3.6.1	2013	71
Drupal	12	4.0.0	2002	7.23	2013	120
phpBB	12	2.0.0	2002	3.0.12	2013	37
MantisBt	8	1.0.0	2006	1.2.15	2013	33
phpMyAdmin	9	2.9.0	2006	4.1.6	2014	129

Fig. 1. Release Statistic for the Examined Projects

Letarte, Gauthier, & Merlo extracted the security model is from PHP source code using a reengineering approach. First a PHP parser is used to extract an intra-procedural control flow graph (CFG) for all the functions of the system. Inter-procedural CFG information is also extracted to represent the conservative calling relationship between functions of the whole system.

The above are the majorly reviewed approaches that how given information this review, which has organized the various aspects of evolution of PHP applications.

3 Software Maturity

In scripting languages a major source of unused code is the employment of third party libraries, which at the same time is an accepted good practice in software development and a possible indication of maturity [2]. In this context, Kyriakakis & Chatzigeorgiou investigated the amount of library code being used over time in each system. Another factor implying software maturity is the stability of the corresponding APIs, and therefore, six classes of possible API changes were examined.

3.1 Library usage

PHP is a rather new programming language and according to Kyriakakis & Chatzigeorgiou has gained popularity during the last decade. An indirect indication of maturity for a given programming language is the development of third party libraries and the employment of them in other projects. In three out of the five projects in our study we have observed a strong trend in using such libraries. As Tulach observes, the trend in modern software development is the use of such pre-made building blocks in order to ease and speed up the development of applications. As we have shown, a side effect is the introduction of unused code blocks, due to the scripting nature of the language. However, the fact that the library's source code becomes part of the system's source code, enables us to measure the ratio of library code over system code, something that is not straightforward with compiled languages.

3.2 Interface stability

The stability of an interface can be characterized by the number and types of changes to the functions' signatures. According to the strict PHP definition, a function signature is only the name of the function, but this does not reflect the interface correctly, since no parameters are included. To track interface changes in more detail, Kyriakakis & Chatzigeorgiou have also considered the mandatory and optional function parameters as well as the default values of the optional parameters. They classified the possible changes to six categories as shown in Figure 2. For each version of the examined systems they have computed the ratio of changes over the total number of signatures, differentiating between the six cases shown in Figure 1. Next, they computed the mean of all versions for each project and the results are summarized in Figure 3. The values for cases C1 to C5 are extremely low, considering the almost ten years of evolution for each project. This fact implies that development teams have paid attention in order not to break backward compatibility and that the corresponding APIs are mature. Changes of the 6th type exhibit a mean ranging from 3.75% for phpMyAdmin, to 14.22% for phpBB, providing further support to the aforementioned claim, since despite the implementation changes for a number of functions, the corresponding signatures remained stable.

	Category	Impact	Severity
C1	Change of mandatory parameters	Breaking function's compatibility, i.e. client has to refactor function invocation	
C2	Addition of optional parameters	Possible extension of function's functionality or enhanced detail in their results. No impact on compatibility, i.e. existing clients do not have to be adapted	
C3	Removal of optional parameters	Possible breaking of function's compatibility: issues to calls that used the removed optional parameters	
C4	Change of default values	Possible breaking of function's compatibility: issues to calls that expected a different value.	
C5	Change of function's return type (identified by PHP annotations)	Possible breaking of function's compatibility: issues to calls that expect different return type.	
C6	Change of function's implementation.	No impact on interface compatibility but a factor that shows interface stability since developers pay attention when evolving a function to keep their interface intact.	

Fig. 2. Change cases of function signatures

Project	C1 (%)	C2 (%)	C3 (%)	C4 (%)	C5 (%)	C6 (%)
Drupal	0.15	0.08	0.03	0.09	0.16	6.02
WordPress	0.06	0.16	0.03	0.27	0.42	6.70
phpBB	0.14	0.35	0.02	3.19	0.97	14.22
MantisBt	0.04	0.11	0.00	0.46	0.50	6.65
phpMyAdmin	0.09	0.09	0.02	0.70	1.26	3.75

Fig. 3. Ratio of changes in function signatures

4 Software Quality

There are many different definitions of quality. For some it is the "capability of a software product to conform to requirements." [7] . The first definition of quality History remembers is from Shewhart in the beginning of 20th century: There are two common aspects of quality: one of them has to do with the consideration of the quality of a thing as an objective reality independent of the existence of man. The other has to do with what we think, feel or sense as a result of the objective reality. In other words, there is a subjective side of quality. According to Juran the word quality has multiple meanings. Two of these meanings dominate the use of the word: 1. Quality consists of those product features which meet the need of customers and thereby provide product satisfaction. 2. Quality consists of freedom from deficiencies. Nevertheless, in a handbook such as this it is convenient to standardize on a short definition of the word quality as "fitness for use".

Following the definition of quality consists of freedom from deficiencies, Kyriakakis & Chatzigeorgiou complement the study with a rather traditional measure, they computed McCabe's cyclomatic complexity (CCN), thereby investigating if PHP practitioners implement comprehensible and thus maintainable code. They calculated CCN per function and then obtained the average CCN of all functions for each version. To make results more readable they categorized the functions according to their CCN in three ranges. A value of 10 is usually considered as a critical threshold [8], [9]. To enable a more fine-grained classification and to comply with critical levels identified by various quality assessment tools, they considered a second threshold at the value of 5. As a result, values in the range [0...5) imply excellent readability, [5-10) medium complexity but still readable code and values higher than 10, code that should be examined closely. Next, we calculated the percentage of functions belonging to each range. The percentages over time are almost constant for all five projects as shown in Figure 4.

5 Adoption of Object orientated Programming (OOP)

Nowadays there is no exact definition of the object-oriented programming (OOP) or the object- oriented programming language. In literature, various authors give a different explanation of these terms. Based on these definitions (Berdonosov, Zhivotova, & Sycheva, 2015), they define the object-oriented programming language as a programming language, basic elements are objects that have their own attributes and methods, and forming a hierarchically organized classes of objects.

According to this definition:

Object is a model (abstraction) of a real essence in a programming system.

Class is an abstract declaration of attributes and methods for a group of similar objects which called instances of class.

Attribute is a parameter declared in a class which characterizes the object (class instance).

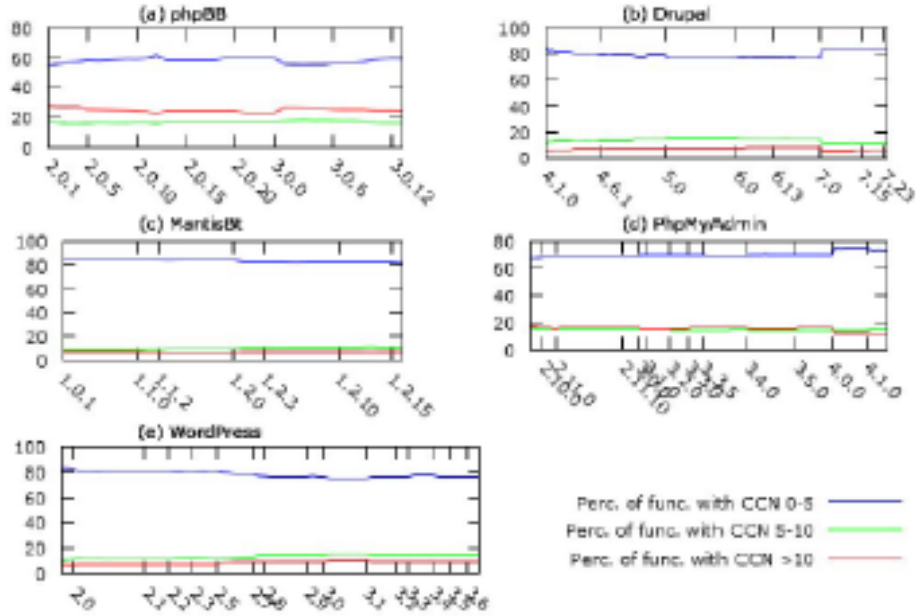

Fig. 4. Evolution of functions in three complexity ranges over time

Method is declared in a class procedure which defines behavior of class instances.

In general, the object-oriented approach to development of programs based on four main mechanisms: abstraction, encapsulation, polymorphism and inheritance.

Abstraction is the process of identifying of the essential characteristics of an object that distinguish it from all other kinds of objects and thus providing crisply defined conceptual boundaries, from the viewpoint of the observer.

Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior.

Polymorphism is the ability of being able to assign a different meaning or usage to something in different contexts and the property of an object respond to a query according to its type.

Inheritance is a mechanism to declare new data types on the basis of existing types in such way that the attributes and methods of the base types become the members of the subtype. [10]

Object orientation in PHP was fully supported in version 5.3, but it was partially supported and used few years before that, staring in early 4.x versions. So there was a period where procedural systems could migrate code to classes [2].

In Figure 5 Kyriakakis & Chatzigeorgiou present the ratio of the number of methods over the total number of functions and class methods of the system code, excluding third party libraries to show the trend of converting the core codebase of the systems to classes. They observe that Drupal after a long period of denial to the object oriented paradigm, even eliminating the small fraction of classes that existed in the early versions, made a turn in version 7.0 with the introduction of classes. The project

with the major change to its coding paradigm is phpBB, where in version 3.0.0 that was a milestone in the project's history, it massively adopted object orientation. WordPress keeps it's slow but steady trend to object orientation, but the huge user contributed code in plug-ins and themes keeps the development team from making major rewrites to the public API of the application. Instead, the developers gradually perform refactoring applications to the internals of the system without breaking backward compatibility. On the other hand, phpMyAdmin that is a widely used project, found in almost any Linux powered web server, has a powerful momentum towards being a fully object oriented system. This is due to the minimal number of user plug-ins or themes, entailing no threat for breaking the public API of the application.

They concluded that migrating applications from procedural to the object oriented paradigm is not only a matter of developers' will or implementation language, but if the project can afford the cost of breaking backward compatibility imposing significant issues to their clients.

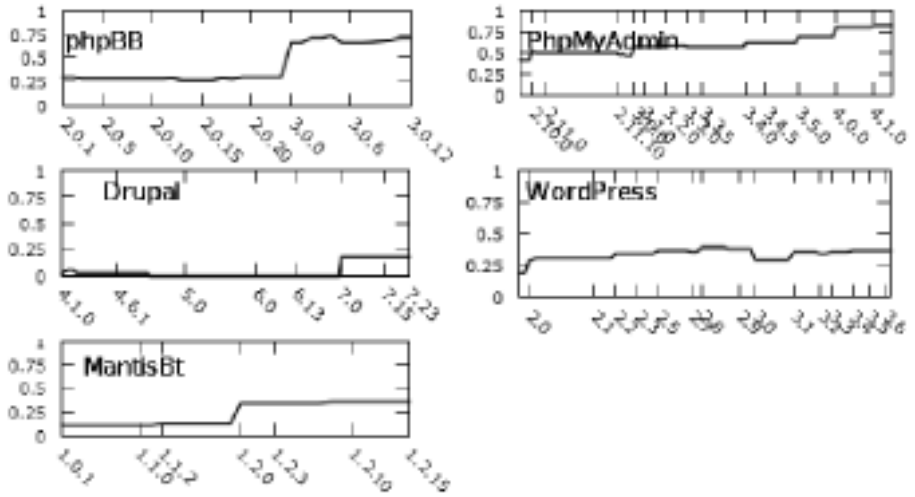

Fig. 5. Methods ratio over total number of functions and methods

6 Security

Web applications require configuration and programming attention to assure security, confidentiality, and trust of the published information. During evolution of Web software, from one version to the next one, security properties may change and possible changes may include new flaws or corrections. Changes to security properties, including access control privileges, can be monitored by observing and analyzing changes between security models extracted from different versions of an application [11].

Property Satisfaction Profiles (PSP), derived from formal security models described by [12], are presented. They are used to investigate the evolution of the secu-

rity models extracted from several versions of a Web application. The motivation comes from the need for observing and comparing security models along the evolution of Web applications. In previous research work by [3], the authors have investigated the evolution of security flaws in Web applications using flow analysis based vulnerability detection [13].

The proposed PSP have been used to monitor security model evolution across several versions of a small PHP open source system, phpBB that implements a bulletin board. Model evolution analysis allows to identify changes in security levels between consecutive versions and may help developers to focus their validation effort on changes at security sensitive statements. Extraction and validation show a linear memory and execution time complexity and are reasonably fast in practice. Required execution time for parsing, extraction of security models and PSP computation take around 44 s. for each individual version and 1349 s. for the 31 versions. Evolution analysis takes 20 s. for all the versions of phpBB. Future research may follow the perspectives of extending experiments to larger and more diversified systems to better assess performance. Also, extensions to the presented approach should be conceived to address more complex security models [11].

6.1 Summary of issues discussed

The overview of the reviewed Topic with the unit of analysis and the conclusions derived based on the findings for each project is provided in Table YES implies that the derived conclusion can be considered as validated for the corresponding project, while a NO implies that the conclusion is not validated.

Table 1. Overview of reviewed Applications

Topic	Conclusions	Wordpress	phpBB	Drupal	MantisBt	PhpMyAdmin	Reference
Software Maturity (Library usage)	Projects reuse code incorporating third party libraries	YES	YES	NO	YES	YES	[2]
Software Maturity (Interface stability)	Function interface remains stable	YES	YES	YES	YES	YES	[2]
Software quality	Complexity remains stable	YES	YES	YES	YES	YES	[2]
Adoption of Object orientated Programming (OOP)	Projects gradually migrate to OOP	YES	YES	YES	YES	YES	[2]
Security	Vulnerability resolution in e.g. SQL injection, Cross-site scripting, Automatic registrations	YES [14]	YES [15]	YES [16]	No	No	

7 Conclusion

The various aspects of the reviews done in this paper are showing how PHP has evolved from a simple web scripting language to a large scale web application and standalone application programming language. The steady fashion in which the sampled PHP applications presented in this review have matured, through the use of third party libraries, having dynamic features presented by Hills & Klint, are an indication PHP has the potential to become one of the biggest programming languages, aside it already dominating the web with over 80% of web applications being powered by PHP [4].

Security being a very crucial part of many web applications, PHP was one of the most vulnarable to attacks such as SQL injection and Cross-site scripting. The adoption of OOP has helped in making PHP applications more secure and easier to maintain. This can be verified from the sampled large and widely used PHP applications in this review. However, it is important to also note that the area of security with regards to php application has had rapid improvement and patching as can be confirmed by [17], With that being a very good indicator, more research has to be done on more invovative security implementations. Further more, the area of PHP Application aging is one other area of research that can give even more information with regards to the evolution of PHP applications.

8 References

[1] M. Hills and P. Klint, "PHP AiR: Analyzing PHP Systems with Rascal," IEEE, pp. 454-457, 2014.

[2] P. Kyriakakis and A. Chatzigeorgiou, "Maintenance Patterns of large-scale PHP Web Applications," IEEE International Conference on Software Maintenance and Evolution, pp. 381-390, 2014.

[3] E. Merlo, D. Letarte and G. Antoniol, "SQL-Injection Security Evolution Analysis in PHP," IEEE, pp. 45-49, 2007. https://doi.org/10.1109/wse.2007.4380243

[4] L. Eshkevar, F. Dos Santos, J. R. Cordy and G. Antoniol, "Are PHP Applications Ready for Hack?," IEEE, pp. 63-72, 2015. https://doi.org/10.1109/saner.2015.7081816

[5] I. HERRAIZ, D. RODRIGUEZ, G. ROBLES and J. M. GONZALEZ-BARAHONA, "The Evolution of the Laws of Software Evolution: A Discussion Based on a Systematic Literature Review," ACM, pp. 28:1-28:28, 2013.

[6] T. Mens and S. Demeyer, Software Evolution, Berlin: Springer, 2007.

[7] "ISO/IEC 9001: Quality management systems -- Requirements," International Organization for Standardization, 1999.

[8] S. Bergmann and S. Priebsch, Real-World Solutions for Developing High-Quality PHP Frameworks and Applications, Wiley, 2011.

[9] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company, 1997.

[10] V. Berdonosov, A. Zhivotova and T. Sycheva, "TRIZ evolution of the Object-Oriented Programming Languages," Electronic Notes in Theoretical Computer Science 314, p. 23–44, 2015.

[11] D. Letarte, F. Gauthier and E. Merlo, "Security Model Evolution of PHP Web Applications," Fourth IEEE International Conference on Software Testing, Verification and Validation, pp. 290-298, 2011. https://doi.org/10.1109/ICST.2011.36
[12] D. Letarte and E. Merlo, "Extraction of inter-procedural simple role privilege models from php code," IEEE Computer Society, p. 187–191, 2009. https://doi.org/10.1109/wcre.2009.32
[13] "Insider and ousider threat-sensitive sql injection vulnerability," in WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006), Washington D.C., 2006.
[14] S. Esser, "Interview with Stefan Esser," 8 November 2016. [Online]. Available: http://blogsecurity.net/wordpress/interview-280607.
[15] Kellanved, "3.0.6 CAPTCHA plugins and you," 7 November 2016. [Online]. Available: https://blog.phpbb.com/2009/06/27/3-0-6-captcha-plugins-and-you/.
[16] D. S. Team, "Security advisories," 8 November 2016. [Online]. Available: https://www.drupal.org/SA-CORE-2014-005.
[17] J. Walden, M. Doyle, R. Lenhof and J. Murray, "Java vs. PHP: Security Implications of Language Choice for Web Applications," in Engineering Secure Software and Systems, Berlin , Springer Heidelberg, 2010, pp. 61-69.
[18] J. Tulach, "Practical API Design: Confessions of a Java Framework," Apress, 2008.
[19] Samoladas, L. Angelis and I. Stamelos, "Survival analysis on the duration of open source projects," Inf. Softw. Technol., vol. 52, no. 9, p. 902–922, 2010. https://doi.org/10.1016/j.infsof.2010.05.001

9 Authors

Douglas Kunda (PHD) is with Mulungushi University, School of Engineering Science and technology, Kabwe, Zambia (dkunda@mu.ac.zm).

Alinaswe Siame is with Mulungushi University, School of Engineering Science and technology, Kabwe, Zambia (alinaswe7@gmail.com).

Article submitted 24 November 2016. Published as resubmitted by the authors 06 January 2017.