A Holistic Evaluation Strategy for the Modernization of Legacy Systems

Meredith A. Barnes and Charmain Cilliers
Department of Computing Sciences
Nelson Mandela Metropolitan University
P.O. Box 77000, Port Elizabeth, 6031
Tel: +2741 – 504 2094, +2741 – 504 2235
email: MeredithAnne.Barnes@nmmu.ac.za; Charmain.Cilliers@nmmu.ac.za

Abstract-Service Oriented Architecture has developed into a current paradigm for business application solutions. This development calls for a need to modernize legacy systems which store the fundamental business procedures for large organisations. The primary goal of this paper is to present a holistic evaluation strategy which can be used to assess the end product of a modernization process applied to a legacy system, specifically a data service in an SOA. The evaluation strategy is comprised of a three-legged approach. The first evaluation leg consists of the analysis of the quality of service of the data service generated by the modernization process. The second leg investigates the effort required by the developer to modernize the original legacy code, depending on the type of modernization approach selected. The third leg measures the efficacy of the data services generated by empirical testing upon completion of the modernization process. The envisaged contribution is the development of an evaluation strategy that ensures successful modernization of legacy systems by assessing each of the major components of the modernization process.

Index Terms – Quality of service (QoS), Software evaluation, Modernization

I. INTRODUCTION

Service Oriented Architecture (SOA) has been identified as an emerging technology that is currently on the rise and will reach maturity within a few years (Figure 1). Consequently, a need for current research in this field has emerged in order to gain enlightenment on the technology.

Legacy systems are used by many organizations because of the data pertinent to the organizations' business processes that are still maintained by them. These legacy systems become outdated after years of use and require more than maintenance to keep them relevant [16]. The modernization of these legacy systems becomes necessary, therefore allowing organizations continued access to their core business functionalities in a more modern architecture.

The current technology climate (Figure 1) implies that a need to modernize legacy systems into web services conforming to SOA requirements will benefit organizations. Services typically consist of a collection of software elements, each of which executes a particular business process [13].

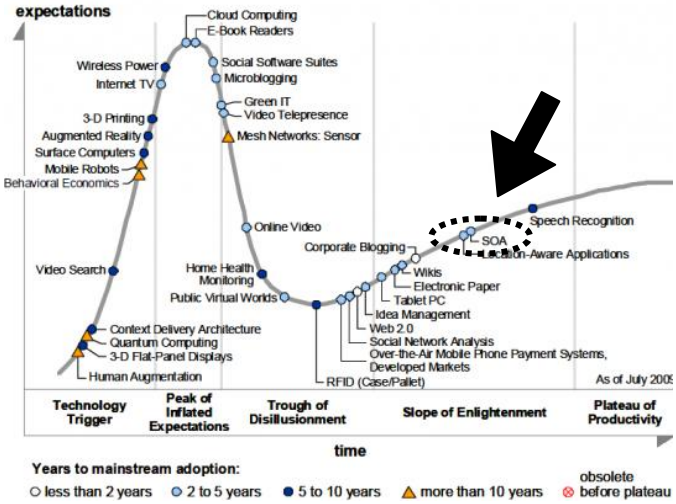

Figure 1. Adapted from Emerging Technologies Hype Cycle [10]

The development of services for an SOA involves adherence to certain protocols and guidelines defined for the specific architecture (Figure 2). The provision of services that provide core business functionality is the responsibility of a Service *Provider*.

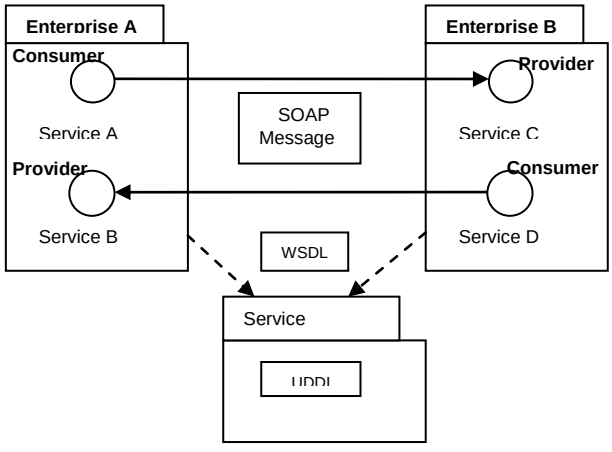

Figure 2. Adapted from "A Sample SOA Environment" [12]

These services are also required by other services or subcontractors, who are Service *Consumers* (Figure 2). Thus the SOA is a combination of service providers and services consumers providing and requesting services, for which all

information is stored, and made accessible to actors in a Service *Registry*. The architecture of an SOA environment depicts the relationship between the service provider, service consumer and service registry (Figure 2). Services share information amongst one another via Simple Object Access Protocol (*SOAP*) messages and information stored in the service registry is based on Universal Description, Discovery and Integration (*UDDI*). The interface between service providers, or consumers, and the service registry can be built based on the Web Service Description Language (*WSDL*) standards.

The functional attributes of services are deciding factors for service consumers when requesting services from service providers [11]. These attributes need to be considered along with non-functional Quality of Service (QoS) attributes. QoS metrics are essential when selecting the best possible execution plan for legacy system modernization with respect to budget and time constraints. QoS metrics define a useful measure for service comparison during the service provisioning process. During the development of services, the effort required by the developer to modernize legacy code to conform to SOA requirements is a useful measure. Software metrics for Object-Oriented (OO) applications are useful for the analysis of the developer's effort to create the service [6].

The modernization of a legacy system into services that perform the same business functionality, but operate in a modern architecture leads to another possible metric for evaluation. Services need to be assessed as a feasible replacement option for the legacy system [5]. An empirical approach can be used to study users' interactions with the services to determine their efficacy [18].

The combination of the QoS evaluation, developer effort evaluation and end product efficacy evaluation leads to a holistic evaluation strategy. The outcomes of the application of this comprehensive evaluation strategy potentially provide results that can be correlated to provide evidence for successful legacy system modernization.

II. BACKGROUND

Understanding the process of and necessity for migrating legacy code to a more modern service-related architecture requires the recognition of concepts such as legacy system modernization and data services. Data is provided as a service, thereby increasing the longevity of the core functionalities of the legacy system.

A. Legacy System Modernization

Since legacy systems contain functionality that is of utmost importance for business longevity, it is essential that they are evolved to integrate with modern platforms and architectures, such as SOA [9]. Modernization covers a broader range of changes to an existing system than the process of system maintenance and is a possible solution for the elongation of a legacy system's lifespan [16]. These changes include restructuring the system, improving system functionality or modifying system attributes. Modernization must, however, conserve a sizable portion of the existing legacy system to conserve the original business rules contained in the system [8]. Modernization approaches typically fall into two distinct categories [9], namely:

- Legacy Integration and Service Enablement; and
- Legacy Transformation.

Legacy integration uses non-invasive wrapping of legacy systems to hide complexity and emphasises modern interfaces to improve interoperability (Figure 3). This category of modernization is used to lengthen the lifetime of legacy systems by exposing the integral functionality of these systems. Consequently, legacy wrapping reduces the cost of integration whilst requiring less immediate planning and design.

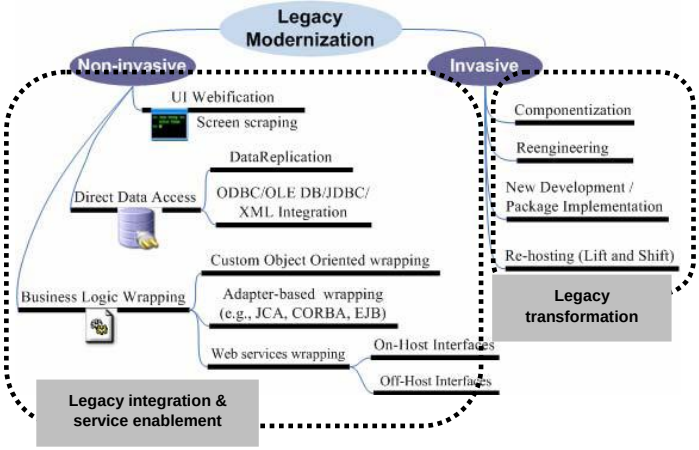

Figure 3. Adapted from "Taxonomy of Legacy Modernization Approaches" [9]

Legacy transformation follows an invasive re-engineering approach to convert legacy systems (Figure 3). A detailed analysis of the existing legacy code and an understanding of the system functionality and data structures lead to the extraction of data definitions and business rules.

Modernization can be classified by the degree of knowledge of system internals required to sustain the modernization approach [8]. Modernization that requires understanding of the functionality and structure of a legacy system is known as white-box modernization, and modernization that only requires knowledge of the legacy system interfaces is called black-box modernization.

White-box modernization requires an initial reverse-engineering process to gain understanding of the legacy system internal structure and operation [8]. This process is called *program understanding* and involves modelling the domain, extracting information from the legacy code and creating representations of the system hierarchy [16]. After program understanding, restructuring of the system defines a transformation from one representation of the system to another at the same level of abstraction [8]. This process typically alters quality attributes of the system including maintenance and performance. White-box modernization is beneficial for simplified future maintenance and extension [9].

Alternatively, black-box modernization is concerned with the evaluation of the inputs and outputs of the legacy system during operation to gain knowledge of the interfaces [16].

Black-box modernization is usually a less complex task than white-box modernization as all it involves is the wrapping of the original system. Wrapping entails encasing the legacy system in a software layer that masks the unnecessary complexities of the legacy code whilst generating a modern interface. The wrapper interfaces with the legacy system during execution of every possible interaction instance [3]. A wrapping approach is termed black-box since after the legacy interface is analysed, the internals are disregarded [16].

There are many advantages for incorporating wrappers into legacy applications [17]. Firstly, the legacy database is not altered in any way. Secondly, the wrapper allows for more functionality to be added to existing legacy applications, such as statistics collection and performance visualization. Lastly, wrappers allow for an incremental modernization process of complex legacy systems. Alternatively, a disadvantage of black-box modernization is the manual recording of interaction scenarios and screen templates [3]. A black-box solution is not always realistic as it sometimes requires knowledge of some system internals by use of white-box techniques [8].

Each of the modernization approaches has benefits in specific domains. However, there is a lack of a comprehensive evaluation of the modernization process that identifies which approach is successful in the modernization of a legacy system into data services.

B. Data Services

Services that provide data in a uniform structure are known as Data Services [4]. These data services have the essential functionality of a distinct business object. Thus, the structure of the data service is determined by and describes the information stored within it for its specific business object type.

Data services have a collection of read methods which are calls to the service to request access to instances of business object types in various ways [2]. Similarly, data services have write methods which allow for the insertion, modification, or deletion of different business object instances. Data services also have navigation methods which are service requests to navigate the relationships between business objects returned via different services.

Importance of quality metrics of services, such as reusability and performance, is stressed for improved service discovery [11]. Several quality attributes are described in existing evaluation strategies that need to be adhered to when generating services for an SOA [1]. These non-functional attributes serve as a guideline for the development of high quality services. Software metrics are used to quantitatively measure quality attributes [6].

III. EVALUATION STRATEGIES

Numerous evaluation strategies exist in the form of guidelines for the development of services for an SOA as well as for the evaluation of the end products of development. These strategies include the adherence to QoS requirements [1], measurement of software metrics to assess code complexity and developer effort [6], and empirical evaluation to gauge the acceptance of the end product with users of the software [18].

A. Quality of Service Attributes

QoS metrics are essential in selecting the best possible execution plan for service discovery and composition within budget and time limitations [11]. A number of quality attributes are relevant to the evaluation of the end product of the modernization process of a legacy system [1, 11], namely:

- Performance;
- Availability;
- Security;
- Testability;
- Interoperability; and
- Reliability.

Performance is measured as the time taken for responses to be generated from requests sent to a service [1], and is indicative of the cost of invoking a service [11]. Availability is defined as a service's ability to be available despite server maintenance or system overload. Security is defined in terms of the confidentiality and protection of data when accessed by a service. A service is required to be testable for modified versions of the service. A service is required to be interoperable on various platforms, the level of interoperability being defined in terms of its reusability. Software metrics can be used to measure reusability of code in a system [6]. The reliability of the service is measured by the amount of system errors encountered and the recovery of the system as a result [11].

B. Object-Oriented Software Metrics

A suite of metrics for object-oriented (OO) code exist based on measurement theory [6]. These metrics were developed to be high level and therefore independent of any programming language. The metrics that will be focused on for the purposes of this study include:

- WMC (Weighted Methods per Class);
- DIT (Depth of Inheritance Tree);
- NOC (Number of Children); and
- CBO (Coupling Between Objects).

The use of these candidate metrics provides insight into the complexity of the code, the effort required by the developer to create and maintain the OO system, as well as effects on quality attributes pertaining to the system.

The effort required by the developer to create and maintain a class is described by the WMC metric [6]. The number of methods per class in an OO system, combined with the complexity of these methods is a clear indication of the amount of time and effort required by the developer to create the class. With regard to the DIT metric, the deeper a class is in a hierarchy of classes, the greater the number of methods it inherits from its hierarchy [6]. This inheritance factor increases the complexity of the class and decreases the ability to predict the behaviour of the class. Alternatively, a greater depth in the inheritance tree implies better reusability of methods.

The NOC metric measures the number of subclasses inheriting the methods of a parent class, namely the scope of its properties [6]. The greater the number of subclasses, the better the reuse of methods from the parent class due to the inheritance property.

The CBO metric provides an indication of the comprehension of the actions of one object on another object due to the former object's calls or references to the latter [6]. If the CBO measure is too high, the reusability and modularity of the class is negatively affected. By minimizing coupling, it is possible to improve maintainability of the code. Thus, reduced changes are required to other sections of the system code.

C. *Efficacy of Data Services*

Many guidelines for the development of high quality services have been suggested in order to assist the developer in the conversion process associated with the modernization of legacy systems. Once the service has been deployed, however, the user requires an effective and efficient means of carrying out the core functionality for which they require the service. Effectiveness is defined as a general goal as to how well a product performs the task for which it has been created [14].

Empirical research in the form of *usability studies* [14] can be used to determine the efficacy of a service, since the service is software with which the user interacts. This strategy is apparent in the case where empirical evaluations are used to support the comparative analysis between an existing legacy system modernization tool and MELIS, a proposed legacy system migration tool [7]. Usability testing is defined as the evaluation approach that involves measuring users' performance and analysing their satisfaction with the system being tested in a formal controlled environment [18].

Quantitative methods are defined as collecting numerical data and analysing it by statistical procedures [15]. Qualitative data is collected as text or images from observations, interviews or questionnaires. Results from statistical analysis of data collected from usability experiments are combined with the participants' feedback to validate the results, seen in similar studies [7].

IV. COMPREHENSIVE EVALUATION STRATEGY

Various evaluation strategies exist and are employed to measure the quality of software produced [1, 5, 6, 11] as well as the satisfaction that the end product provides the customer [18]. A comprehensive evaluation strategy is proposed. The approach evaluates the design, implementation and outputs of a modernization process to aid in the assurance of a thorough modernization process and the precise performance of the outputs after modernization [5]. The holistic modernization evaluation strategy is comprised of three different criteria:

1. QoS attributes of the services generated;
2. Developer effort required to modernize legacy system; and
3. Efficacy evaluation of the service generated.

To validate this evaluation strategy, the three evaluation legs will be applied to a case study. The legacy system in the case study allows student assistants at a university to maintain their information, select courses to assist and mark their attendance for sessions in the courses they assist. This legacy system went through a white-box modernization process to expose the data as a service, using the design principles shown in similar studies [2]. The data services were created to retrieve data from the database with read methods such as *getStudentInfo* and *fillDetails* (Table 1). Data services also contained write methods such as *updateDemi* (Table 1).

A consolidated list of QoS attributes can be drawn from combining those presented (Section III). The most appropriate and frequently used QoS metrics are used as a guideline for service generation from the modernization approach [1, 2, 11]:

1. Performance;
2. Reliability;
3. Interoperability; and
4. Reusability.

Testability is excluded from the four final attributes as it is identified as a minor quality attribute [11]. Security and availability are excluded from the elected QoS metrics due to the nature of the case study. The student assistant system operates within a closed network where data security is not a necessity. The availability of the services despite server overload is not a concern in this study as it is not likely that the data services would be queried simultaneously by several users.

Service Methods	Mean Time (s)	Std Dev
checkStudent	0.417	0.051
getStudentInfo	0.045	0.002
fillNoticeBoard1	0.045	0.005
fillNoticeBoard2	0.074	0.005
getMessages	0.044	0.002
fillDetails	0.062	0.005
updateDemi	0.067	0.009
getSessionTableData	0.069	0.026
getSessionColumnData	0.030	0.003
getHoursTableData	0.064	0.004
getHoursColumnData	0.035	0.006

Table 1. Mean Response Times for Service Calls

Performance of the data services created during modernization was measured in seconds as each service method was called by the client application. The service methods were tested by seven participants and the mean and standard deviations were recorded (Table 1). Considering that the services are called with a SOAP request, perform a SQL query to retrieve data and finally return the data via a SOAP response to the client, the performance results are consistently low, with the exception of the *checkStudent* method. This method is called first during login to the system and the connection to the data source is first made here, explaining the excess time taken for the service to respond. Participants noted that the response times of the data services were no different than the performance of the legacy system (n = 3).

Reliability of the data sent to and retrieved by the services was dealt with in the service client. Checks are performed on all data to ensure that it is meaningful and the system provides users with responsive feedback for the actions which they perform. Finally, the interoperability of the services describes the reusability of the services on any platform. To ensure reusability of the services generated from modernization, the services were written with Java and deployed on a Glassfish application server using SOAP standards. The nature of SOAP services is inherently platform independent due to the exchange of messages in SOAP format using XML. The service client classes maintain the states of the interface during the application's execution (Table 2), whilst the data services contain the methods to access and modify data and the service methods are called by the service client classes as needed.

	Home	Details	PrintSession	PrintHours	SessionModel	HoursModel
WMC	22	22	11	16	8	7
# Methods	11	6	6	8	6	6
DIT	0	0	0	1	1	1
NOC	0	0	0	0	0	0
CBO	2	1	2	1	0	0

Table 2. Metrics for Classes in the Service Client

The DIT and NOC metrics are useful measures for the verification of code reusability. The calculation of the depth of a class in a class hierarchy reveals the level of method reuse due to inheritance [6]. The reusability of code in the client application classes is low; however, low DIT and NOC values also indicate reduced complexity and increased prediction of behaviour of the classes (Table 2). The CBO metrics collected also prove useful in determining the reusability of classes of code in the application. The *Home* and *PrintSession* classes show the highest values for coupling, thus increasing their reusability slightly.

The WMC metric provides a measure of effort required by the developer to create the data services from the legacy code during modernization. Thus, the number of methods per class and their perceived complexity provides an indication of the time and effort to create and maintain the code [6]. The WMC metrics for the classes as well as the number of methods per class are indicated in Table 2. The complexities of the methods are relatively low amongst the classes in comparison to the number of methods per class, thus improving the quality of the system. This reduced complexity of the service client is due to the incorporation of data services for any data retrieval or insertion (Table 1).

Reverse-engineering rules applied to legacy systems require validation by empirical research [5]. The modernization of legacy systems of various designs produces various end results. Empirical research performed in a similar environment to which the results will be applied with realistic subjects is relevant [15]. Admittedly, this form of research may be time-consuming and costly, but the industrial benefit lies in the implication of the results. After modernization and software metric evaluations were completed, a usability study was performed to obtain quantitative and qualitative feedback from representative users of the system. To identify any usability issues, a sample of seven participants was selected to perform three tasks on the modernized student assistant service. The participants consisted of six male and one female student assistant, who were familiar with the legacy system as they had used it at least once a week previously (n = 7). Participants' times taken per task were recorded to assess the efficiency of the data services. The times per task were consistently low for all tasks, at less than one minute (Table 3). The mean times per task are an indication of the efficiency of the services to provide timely responses to the users.

	Mean Time (s)	Std Dev
Task 1	38.51	7.14
Task 2	20.36	7.15
Task 3	35.96	10.83

Table 3. Participants Mean Times per Task

	Frequency
Task 1	0
Task 2	0
Task 3	5

Table 4. Error Rates per Task

The effectiveness of the system to provide users with the data they requested was assessed by retrieving task success and task completion measures. All participants were able to complete each of the three tasks, although task three produced high error rates (Table 4). Whilst executing task three, participants struggled to navigate to the interface to print the total hours of service. During modernization, the interface was directly translated from the legacy system, thus indicating design flaw in the original system as well. A recommendation from this observation is to improve the visibility of the option to print the total hours of service, thus improving navigability of the modernized system.

Lastly, a post-test System Usability Scale (SUS) was performed to determine system acceptance as a whole by participants, identified by unique participant ID's (Table 5). The scores for each of the 10 questions were normalized and collated into a final score.

	P01	P02	P03	P04	P05	P06	P07	Mean
Score	92.5	92.5	90	100	82.5	75	77.5	87.14

Table 5. SUS Scores per Participant

Consistently high scores were given by each of the participants, indicating a high acceptance rate for the modernized system. Combined with encouraging user comments and the system's efficiency and effectiveness to provide users with the data they requested, it can be seen that the data services are a feasible replacement for the legacy system.

V. CONCLUSION

Due to the growing enlightenment on SOA technologies [10], there is an opportunity for organisations to use these emerging technologies as a platform to modernize legacy systems. The benefit of the modernization process is the

lengthened lifespan of core business processes within existing legacy systems. Once data services have been generated by modernization, it is necessary to evaluate these services. A holistic evaluation strategy for legacy system modernization consisting of three evaluation legs is proposed. Firstly, QoS attributes are defined as a guideline for development of data services. The second leg involves the prediction effort required by the developer to generate the data services during modernization. Thirdly, the data services must be evaluated by empirical studies to gauge performance of the service as well as user satisfaction.

Existing evaluation strategies include the service's adherence to QoS guidelines by the identification of major quality attributes of services for an SOA [11]. Also, the use of software metrics during the development process to ensure software of a high quality is identified [6]. Lastly, user studies have been performed to assess performance metrics as well as gain feedback regarding qualitative measures of software end products [7]. By combining these existing evaluation strategies to produce a comprehensive evaluation framework, the overall success of a modernization approach can be gauged.

The application of this evaluation approach to the case study in this paper demonstrated how to use QoS metrics as a guideline for the development of high quality data services. The extent to which the guidelines were met was measured by the use of software metrics (Section IV). These analytical metrics validated the proper design of the modernized data services. These metrics alone, however, don't give an overall view of the usefulness of the new system. To measure the effectiveness of the data services, usability tests were conducted. The results obtained from the usability tests indicated a high acceptance rate by participants. The combination of the analytical evaluation results and the usability testing results illustrate the success of the modernization approach applied to the case study.

Perceived benefits of this combined modernization evaluation approach include the collection of data throughout all phases of development. Guidelines have been recognised that need to be adhered to. These guidelines provide rules for the development of high quality services. Software metrics provide a quantitative measure for the evaluation of the modernization outputs. The developer's effort measure could be beneficial in the determination of which modernization approach is more suitable for the migration of legacy code to data services. Lastly, user studies of the services generated allow for the assessment of the final product's efficacy and the user's satisfaction. This holistic evaluation approach aims to ensure the successful outcome of legacy system modernization.

The comprehensive evaluation strategy was applied to a white-box modernization case study. The results from this pilot study were feasible, showing that the modernization of the legacy system into data services was an efficient and acceptable process. The analytical evaluation of the software metrics and quality of the service also yielded positive results regarding the design of the data services. The success of the modernization effort has served as a validation of the selection of the criteria for this evaluation strategy.

VI. REFERENCES

[1] Bianco, P., Kotermanski, R. & Merson, P. (2007). Evaluating a Service-Oriented Architecture. Carnegie Mellon University.

[2] Borkar, V., Carey, M., Lychagin, D., Westmann, T., Engovatov, D. & Onose, N. (2006). Query Processing in the AquaLogic Data Services Platform. International Conference on Very Large Data Bases (VLDB'06).

[3] Canfora, G., Fasolino, A., Frattolillo, G. & Tramontana, P. (2006). Migrating Interactive Legacy Systems to Web Services. *Conference on Software Maintenance and Reengineering.*

[4] Carey, M. (2006). Data Delivery in a Service-Oriented World: The BEA AquaLogic Data Services Platform. International Conference on Management of Data.

[5] Chiang, R.H.L., Barron, T.M., Storey, V.C. (1997). A Framework for the Design and Evaluation of Reverse Engineering Methods for Relational Databases. *Data and Knowledge Engineering*, **21**, 57-77.

[6] Chidamber, S., Kemerer, C. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, **20**, 476 – 493.

[7] Colosimo, M., De Lucia, A., Scanniello, G. & Tortora, G. (2008). Evaluating legacy system migration technologies through empirical studies. *Information and Software Technology,* 51**,** 433-447.

[8] Comella-Dorda, S., Seacord, R.C., Wallnau, K., Robert, J. (2000). A Survey of Black-Box Modernization Approaches for Information Systems. *International Conference on Software Maintenance*.

[9] Erradi, A., Anand, S., Kulkarni, N. (2006). Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture. *International Conference on Services Computing*.

[10] Gartner Group. (2009). Hype Cycle for Emerging Technologies, 21 July 2009.

[11] Jeong, B., Hyunbo, C., Choonghyun, L. (2009). On the Functional Quality of Service (FQoS) to Discover and Compose Interoperable Web Services. *Expert Systems with Applications,* 36, 5411-5418.

[12] Liegl, P. (2007). The Strategic Impact of Service Oriented Architecture. *International Conference and Workshops on the Engineering of Computer-Based Systems.*

[13] Sanders, D., Hamilton, J. & Macdonald, R. (2008). Supporting a Service-Oriented Architecture. *Spring Simulation Multiconference.*

[14] Sharp, H., Rogers, Y., Preece, J. (2007). Interaction Design: Beyond Human-Computer Interaction, John Wiley & Sons.

[15] Sjøberg, D.I.G., Dybå, T., Jørgensen, M. (2007). The Future of Empirical Methods in Software Engineering Research. *Future of Software Engineering (FOSE '07).*

[16] Seacord, R. C., Plakosh, D. & Lewis, G. (2003). Modernizing Legacy Systems Software Technologies, Engineering Processes, and Business Practices, Addison-Wesley.

[17] Thiran, P., Hainaut, J., Houben, G., Benslimane, D., (2006) Wrapper-based Evolution of Legacy Information Systems. *ACM Transactions on Software Engineering and Methodology*. 329-359.

[18] Tullis, T., Albert, B. (2008). Measuring the User Experience, Morgan Kaufmann.

Meredith A. Barnes received her BSc and BSc (Hons) in Computer Science and Applied Mathematics from Nelson Mandela Metropolitan University (NMMU). She is currently pursuing a MSc in Computing Sciences at NMMU.