Comparing the Impact of Service-Oriented and Object-Oriented Paradigms on the Structural Properties of Software

Mikhail Perepletchikov, Caspar Ryan, and Keith Frampton

RMIT University, School of Computer Science and Informational Technology
{mikhailp, caspar, keithf}@cs.rmit.edu.au

Abstract. Service-Oriented Architecture (SOA) is a promising approach for developing enterprise applications. While the concept of SOA has been described in research and industry literature, the techniques for determining optimal granularity of services and encapsulating business logic in software are unclear. This paper explores this problem using a case study developed with two contrasting approaches to building enterprise applications that utilise services, where one of the approaches employs coarse-grained services developed based on the principles of Object-Orientation (OO), and another approach is based on embedding business rules and logic into executable BPEL scripts and constructing a system as a set of fine-grained services. The quantitative comparison based on a set of mature software engineering metrics showed that a system developed using the BPEL-based approach has a potentially higher structural complexity, but at the same time lower coupling between software modules compared to an OO approach. It was also shown that some of the existing software metrics are inapplicable to SOA, hence new metrics need to be developed.

1 Introduction

Service-Oriented Architecture (SOA) is an approach for constructing integrated enterprise software systems that employ services, where a service represents a function that is self-contained, and does not depend on the context or state of other services [7]. SOA-based systems are defined as a collection of interacting services that offer well-defined interfaces to their potential users. One of the driving factors behind SOA is its business alignment [2, 17]. Businesses depend on information technology for their everyday tasks, and as such, the logic and rules that drive the business are integral part of software. The traditional approach is to code business logic into software itself, whereas SOA in conjunction with Business Process Modelling (BPM) allows situating business logic within executable business processes that can be designed and implemented by business modelers with the aid of tool support, thus providing a higher level of abstraction for encapsulating business logic, and facilitating reconfiguration.

Although various publications have described benefits of embedding business logic into executable business processes, including increased maintainability and reusability of software [5, 14], such descriptions have not been empirically evaluated or supported by case studies. Therefore, this paper provides an initial quantitative evaluation of the impact of embedding business logic into business processes on the structural software attributes of size, complexity, coupling and cohesion.

R. Meersman et al. (Eds.): OTM Workshops 2005, LNCS 3762, pp. 431 – 441, 2005.

The contribution of the paper is as follows. A case study was used to illustrate issues related to granularity of services in service-oriented development in order to drive further research in the area of service granularity and implementation of business logic in software with the aim of specifying guidelines applicable to SOA development. To facilitate this case study, a prototypical enterprise application was designed and implemented using two contrasting approaches. The approaches represent two extremes in service granularity, where one of the approaches is based on the principles of Object-Orientation (OO), consisting of one coarse-grained service with business logic embedded into a hierarchical OO design structure, and another approach based on embedding business rules and logic into executable BPEL4WS scripts and constructing a system as a set of fine-grained services. By investigating two extremes in service granularity this paper explores issues related to coarse and fine grained services with the aim of finding a right balance between the extremes in granularity since in practice neither of them is ideal. Note that a fine grained OO-based design and a coarse grained BPEL-based design are also possible in theory, and as such, may be investigated in future work.

The impact of these approaches on the structural software attributes of size, complexity, coupling, and cohesion was quantitatively measured by applying a set of eight established software engineering metrics to the resulting designs and prototypical implementations. Thus, the applicability of some of the existing metrics to SOA was indirectly evaluated. Finally, the initial systems were extended and measured in order to evaluate changes in both approaches using the same metrics.

The results show that systems developed using the OO-based approach exhibit higher inter-module coupling, but at the same time have potentially lower structural complexity. Furthermore, there is a need for new metrics specifically tailored to SOA since the existing metrics were not immediately applicable to the BPEL-based approach. Such metrics will be derived and evaluated in future work.

The rest of the paper is organised as follows: Section 2 presents background material including discussion of the concepts of SOA and BPM, and a short description of the software attributes under investigation. Section 3 provides a detailed description of the case study including methodology, metrics used to quantify the designs/implementations, and illustration of the actual designs. The metrics based evaluation of the case study is presented and analysed in Section 4. Finally, Section 5 closes with conclusions and a discussion of future work.

2 Background

This section serves two purposes. Firstly, it briefly describes key concepts of Service-Oriented Architecture (SOA) and Business Process Modelling (BPM). Secondly, it presents the structural software attributes being investigated.

2.1 Characteristics of SOA

SOA is an abstract concept of how software services should be composed and orchestrated. A conceptual model of SOA consists of two primary parties: a service provider, who publishes a service description and realises the service; and a service consumer, who finds the service description in a registry and invokes the service [2].

The notion of a service is similar to that of a component, in that services, much like components, are independent building blocks that collectively represent an application. However, services are coarser grained than components; and they should exhibit complete autonomy from other services, meaning that each service should be implemented separately from other services resulting in a loosely coupled system [7].

For the purpose of this paper, SOA is defined as a *software development paradigm based on the concept of encapsulating application logic within independent, loosely coupled, stateless services that interact via messages using standard communication protocols, and can be orchestrated using business process languages*. This particular definition was chosen since it captures the main essence of SOA from both, representational (architectural) and development perspectives.

In SOA, instead of thinking of services as interfaces to software functionality that connect to other interfaces, organisations should consider services as enablers of business processes. Technically, a business process in SOA is a service; hence developers can compose local or external services into executable business processes that are exposed externally as services.

2.2 Business Process Modelling

Business processes reflect workflows within and between organisations. Business process modelling (BPM) describes activities that interact with various intra/inter-organisational elements while supporting the operation of the business.

There are a large number of techniques proposed for business process modelling ranging from flow charts and workflow languages to UML and Petri Nets, each having various supporting business process languages. Until recently, Microsoft used the Pi-Calcius model with XLANG (http://msdn.microsoft.com/library/en-us/biztalks/), IBM used Petri Nets with Web Services Workflow Language (WSFL) (http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf), and BPMI.org developed the Business Process Modelling Language (BPML) (http://www.bpmi.org/_vti_bin/shtml.exe/bpml-spec.htm). Such languages allow business process models to be designed and directly executed via middleware support.

The Business Process Execution Language for Web Services (BPEL) [1] is the latest in the series of BPM languages, uniting the ideas from the XLANG and WSFL specifications. It is arguably the most widely used language since it was developed by a consortium of major software vendors (such as IBM, Microsoft, and BEA). BPEL is currently in the standardisation stage at OASIS (http://www.oasis-open.org/), having adopted a model that is similar to the one promoted by BPML, thus facilitating a convergence of standards in the industry [14].

2.3 Structural Software Attributes

A software attribute of a product is any feature or property of the product. The attributes used in this paper are structural design attributes of size, complexity, coupling and cohesion. In line with its common usage [6, 8], *coupling* is defined as a measure of the extent to which interdependencies exist between software modules[1]. *Cohesion* is defined as the extent to which elements of a module contribute to one and only one

[1] A *module* is any software entity including a class, service, or business process model.

task. *Complexity* is defined in terms of the internal work performed by a service. Finally, *size* refers to the size of a software product in terms of lines of code.

In general, low coupling and complexity, and high cohesion are desired; however coupling and cohesion are conflicting requirements that usually require compromise. Structural attributes of service-oriented software (SO) have a causal impact on external quality attributes as described by Perepletchikov et al. [15].

3 Case Study

The case study described in this section and evaluated in Section 4 investigates the impact of service-oriented and object-oriented approaches to SOA development on the structural software attributes of size, complexity, coupling and cohesion. For the purpose of the case study, a portion of an Academic Management System (AMS) was designed and implemented. Such a system is used in tertiary education institutes to administer students and staff. The business logic and rules correspond to those used at RMIT University (Melbourne, Australia). The AMS is an inter-organisational system since it communicates with external services provided by Australian Tax Office (ATO), Department of Immigration, and the University's Library[2], and thus is suitable for SOA-based case study.

The following top-level use-cases were identified during the Analysis phase: enroll student, withdraw student, allocate student to tutorial group, facilitate fee payment, obtain details, update results, change workload, and allocate staff tasks. The use-cases were ranked based on their importance. The focus of this case study is on the *enroll student* use-case, which is the most important use-case in the system according to the rankings. This use-case was designed and implemented first, thus conforming to the approach prescribed by Rational Unified Process (RUP)[12] where systems are developed iteratively in a number of increments with the most important parts of the system being developed first based on the ranking of use-cases.

For each use-case in the system a workflow was created by university administrators who have knowledge of business logic and rules required for fulfillment of each use-case. For example, the following is informal description of the *enroll student* workflow associated with the *enroll student* use-case:

- The student submits an enrollment application specifying the course he/she wishes to enroll into.
- An administrator checks that student does not have any outstanding loans with the library.
- If there are no outstanding loans, the administrator checks a number of constraints (business rules) in order to determine whether this student should be allowed to enroll into the desired course. The constraints vary based on the student type, where a student can be either International (INT) or Local (LOC), and Undergraduate (UG) or Postgraduate (PG). Each combination of such types has different business rules associated with enrollment.

[2] Example web services located on the machines other than the machine AMS was running on were used for the purpose of this case study.

- If based on the business rules the student should be allowed to enroll into the course, an administrator will request approval from the appropriate supervisor for that course.
- When the enrollment is approved, an administrator will organise a payment for the additional course depending on the student type and preferred payment method, and will finally update student data to capture the changes associated with the enrollment.
- The student gets notified about the outcome of enrollment application, the workflow ends.

3.1 Methodology

The evaluation of the effect of service granularity on structural software attributes was performed in three stages: i) firstly, *enroll student* workflow was partially[3] developed using two contrasting approaches, and a set of software metrics (the metrics are described in the following sub-section) was applied to the resulting designs and implementations in order to evaluate the impact of both approaches on the structural software attributes of size, complexity, coupling, and cohesion; ii) next, the original business rules used in the *enroll student* workflow were modified. The modifications were propagated to the designs and implementations, and the modified systems were re-measured using the same metrics; iii) finally, another use-case (*withdraw student*) was developed and integrated into the existing system. Again, the modified systems were re-measured using metrics. Conducting the evaluation in three stages allowed establishing trends related to system changes.

The first development approach was based on the principles of OO [11], consisting of one coarse-grained *enroll student* service with business logic embedded into the actual software implementation. From this point onward, this approach will be referred to as the *Object-Oriented (OO) approach*.

The second approach involved embedding business rules and logic into '*enroll student'* process, and constructing a system as a set of fine-grained services. In this approach, the actual implementation was done mainly in the process itself. This approach will be referred to as the *Service-Oriented (SO) approach*.

3.2 Metrics

Since specific metrics for measuring software attributes of SOA-based systems are yet to be defined, one of the objectives of this work was to assess the applicability of conventional software engineering metrics to SOA. A set of eight well-established metrics was chosen based on their applicability to both the OO and the procedural BPEL approaches. The decision was made to use a set of OO-specific CK (Chidamber-Kemerer) metrics [4] together with the traditional McCabes Cyclomatic Complexity (CC) [13] and Lines of Code (LOC) [8] metrics. The CK metrics consist of Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response For a Class (RFC), Weighted Methods per Class (WMC) and Lack

[3] Systems are not fully operational due to the absence of a data persistence layer, although the designs and implementations are structurally complete.

of Cohesion of Methods (LCOM). These metrics were chosen since they are well-established and highly-referenced in the research literature [3, 4, 8, 10, 16].

For the purpose of this study the metrics usage is as follows: i) the LOC measure was used to compare the *size*; ii) the CC, DIT, NOC, and WMC metrics were used to compare the *complexity*; iii) the CBO and RFC metrics were used to compare the *coupling* between classes in the OO approach, and coupling between business processes and services in the SO approach; and iv) the LCOM metric was used to compare module *cohesion* of classes, services, and business processes.

3.3 Base Case

Figures 1 and 2[4] show the static and dynamic aspects of the design developed using the *OO approach*. This design makes appropriate usage of OO constructs such as inheritance, aggregation, and association. Given that the problem of enrolling students is hierarchical by its nature since there are a number of different types of students, each with their own prerequisites and constraints; OO provides a well-documented approach for this problem [9, 11]. Such an approach is based on specialisation, and polymorphic deferral of the enrollment procedure to a particular subclass of a student during a run time, with business logic and rules defined in the `enroll()` method of a particular student type. The Java programming language was used for implementation, and the interface (wsdl file) was generated using Apache Axis 2.0 technology (http://ws.apache.org/axis2/).

The business process model representing the *enroll student* workflow is a core of the *SO approach*, where a process itself explicitly describes all the business logic and rules as shown in Figure 3[4]. The services invoked from this process are finer grained than OO counterparts performing basic operations based on data manipulation. BPEL was chosen as a modelling/execution language for the *'enroll student'* process since it is most-widely used business process language as was described earlier. For design, implementation, and deployment purposes, the Oracle BPEL Process Manager 2.1 and BPEL Designer 1.0 (http://www.oracle.com/technology/products/ias/bpel/index.html) tools were used. The services were implemented using Java programming language.

3.4 Modifying Business Logic

After the initial systems were measured, the business rules were modified to measure changes required when altering business logic. Originally, prior to enrolling student into a course, a system had to check whether the addition of a course is within the allowed load limits based on the combination of two student roles, career level (UG or PG) and residency status (LOC or INT). The modifications introduced an additional role to indicate whether the student is Part-Time (PT) or Full-Time (FT), with the constraint that INT students must be FT according to Federal Government regulations.

[4] The diagrams related to the case study can be found in the online appendix at http://www.cs.rmit.edu.au/~mikhailp/research/MIOS/appendix.pdf, which is also available from the authors on request.

Implementing these changes using the *OO approach* involved adding two interfaces for each new type of the student (`PTStudentInterface` and `FTStudentInterface`); and sub-classing `LocalUGStudent` and `LocalPGStudent` classes resulting in four extra classes as shown in Figure 1 in the online appendix[4] (shaded out section). To modify the BPEL version, the actual process was changed in order to facilitate the modified requirements. Due to the space limitations the resulting BPEL process is not shown in this paper, it can be found in Figure 4[4]. The impact of the changes on both designs in terms of structural attributes is quantified in Section 4.

3.5 Adding Additional Functionality

To determine the changes required when adding extra functionality to the system, the *withdraw student* use-case was designed and integrated using both approaches.

The implementation of this use-case in the *OO approach* required only the addition of the `withdraw()` method to the `StudentMangementService` and `StudentInterface` as shown in Figure 1[4] (circled out) and implementation of this method in the appropriate student sub-classes. The *SO approach* required developing a new '*withdraw student*' business process, as well as the addition of new operations to the services. The developed workflow is shown in Figure 5[4].

4 Comparison of the Development Approaches

The measures collected from the designs and implementations of partial AMS system are shown in Table 1. The measures were collected in three stages: after the original *enroll student* use-case was developed; after the *enroll student* use-case was modified; and after the addition of *withdraw student* use-case.

Table 1. Metrics collected by measuring designs/implementations of partial AMS

	Size	**Complexity**				**Coupling**		**Cohesion**
	LOC	***CC***	***WMC***	***DIT***	***NOC***	***CBO***	***RFC***	***LCOM***
Enroll Student - *Original*								
OO	950	16	60	4	4	14	30	-
SO	990	14	16	1	1	8	10	-
Enroll Student - *Modified*								
OO	1280	28	62	5	6	20	**36**	-
SO	1060	***19***	18	1	1	8	12	-
Enroll Student (*Modified*) and Withdraw Student								
OO	1410	30	65	5	6	20	**70**	-
SO	1440	***30***	20	1	1	16	22	-

4.1 Metrics Collection Process

Since the CK metrics were designed specifically for object-oriented systems, they are not immediately applicable to the *SO approach* due to a lack of inheritance and

aggregation. Therefore, the following assumption was made to facilitate measurement of BPEL processes and services using CK metrics:

Business Process (BP) = Service = Class

As such, the following procedures were used to measure software attributes of designs and implementations developed using both approaches:

- **Lines of Code (LOC):** i) in the *OO approach,* all java source files (excluding comments) were counted as code; ii) in *SO approach*, bpel scripts and the actual services (implemented in Java) were counted as code.
- **Cyclomatic Complexity (CC):** i) in the *OO approach*, all conditional statements and loops within implementations of method bodies were counted in order to derive CC according to [13]; ii) in the *SO approach,* all conditional statements and loops within *BP* and individual services were counted in order to derive CC again according to [13].
- **Weighted Methods per Class (WMC):** i) The WMC can be measured by either counting methods implemented within a class, or finding total CC of the methods [16]. Since the CC was already measured, the total number of methods in the system was calculated to indicate WMC in the *OO approach,* and the total number of methods in service implementations indicates WMC in the *SO approach*.
- **Depth of Inheritance Tree (DIT) and Number of Children (NOC):** i) in the *OO approach*, DIT and NOC were calculated according to [4]; ii) in the *SO approach,* DIT and NOC should always have a count of one due to an absence of inheritance in SOA. Therefore, the observation can be made that inheritance-related metrics are inappropriate for measuring SO approach since the concepts of SOA do not directly map to the concepts of OO.
- **Coupling Between Objects (CBO):** i) In the *OO approach*, the inheritance was taken into consideration when measuring CBO according to [3], if a method call is polymorphic all the classes to which the call can go are included in the coupled count. The decision was made to measure the classes included in the *enroll/withdraw student* sequence diagrams only, and then select the class with the highest CBO count for the comparison purposes (the `Student` class was selected). This decision was based on the fact that it would be inappropriate to calculate means or variances in CBO counts since there are only one (or two, in the extended case) *BP*s in the *SOA approach; ii)* In the *SO approach*, a coupling of business processes to services was measured, where a business process is said to be coupled to a service if one of them sends the message to the other.
- **Response For a Class (RFC):** i) In the *OO approach*, the RFC count includes all methods accessible within the class hierarchy and was measured according to [3, 4]. Only the response sets for the `enrollStudent()` and `withdrawStudent()` methods in the `StudentManagementService` class were counted to allow for objective comparison since the *SO approach* was based purely on these two functionalities; ii) in the *SO approach*, RFC count indicates the number of messages sent from a given business process to the associated internal/external services.
- **Lack of Cohesion of Methods (LCOM):** i) In the *OO approach*, the systems were not fully implemented thus LCOM could not be measured; ii) It was discovered that LCOM is not appropriate for the *SO approach* since business process has only one method, and services have functional nature, therefore LCOM cannot be calculated due to a lack of service (class) variables.

Note that only the direct designs/implementations artifacts were measured; Java libraries, etc. were not used.

4.2 Discussion

Prior to measuring the resulting designs and implementations, three *informal* hypotheses were defined based on a critical analysis of related literature [2, 7, 14, 17] and the authors' practical experience with SOA development:

- ***H1*:** the designs/implementations developed using *SO approach* exhibit *lower coupling* compared to those developed with *OO approach* since there are no 'strong' relationships between services and business processes in SOA-based systems.
- ***H2*:** the *SO approach* allows for *easier propagation of changes* in business logic compared to the *OO approach* since the logic is encapsulated in BPEL scripts rather than application code.
- ***H3*:** The designs/implementations developed using *OO approach* exhibit *lower complexity* than the ones developed using *SO approach* since the OO paradigm has advantages of being mature and well-established [9, 11], it can solve various problems using high-level design abstractions such as design patterns [9].

A number of general observations can be made concerning the results of the case study in regards to the above-described informal hypotheses. Firstly, the CBO and RFC counts are higher for *OO* than *SO approach*, and the count for RFC increases rapidly in OO design when a new functionality is added to the system as shown in Table 1 (highlighted in bold) showing that SOA introduces lower coupling compared to traditional approaches such as OO as was expected (***H1***).

Secondly, in *OO approach* there is a large increase in CC and LOC when business logic is modified compared to the increase in CC and LOC for *SO approach* as shown in Table 1 (underlined). As such, the observation can be made that the propagation of changes is easier in *SO approach* ***(H2)*** for this particular change.

Finally, although the measures of CC for *SO* are roughly equal to those for *OO*, there is a potentially large explosion of complexity in *SO approach* as shown in Table 1 (highlighted in bold italic). This is due to the fact that a new business process will be developed every time a new functionally is added to the system. Even though the *OO approach* has a larger count for WMC, this measure does not reflect true complexity since most of the methods counted as part of WMC in *OO approach* are accessors and mutators. Hence, a prediction can be made that CC for the fully implemented system in SO will be significantly larger than that for OO, thus supporting ***H3***.

Furthermore, in the process of obtaining measures it was noted that DIT, NOC, and LCOM metrics do not provide proper measure for attributes under investigation in the *SO approach* as explained in the previous section. Also, CC in isolation does not fully indicate complexity of a business process since it does not take into account types of BPEL constructs and interface complexities of associated services. Hence, there is a need for a set of metrics that are specifically suited for measuring structural properties of SOA-based systems. For example, by assigning particular weights to each type of BPEL activity, a complexity of business process could be measured.

4.3 Limitations

There are a number of limitations associated with the case study. Firstly, only *enroll student* and *withdraw student* use-cases were designed and implemented. Secondly, implementations are not fully operational due to the absence of a data persistence layer, although the designs and implementations are structurally complete. Such factors could influence structural attributes under investigation. Furthermore, there are many different ways of designing AMS using both *OO* and *SO approaches*, as such, not all designs will exhibit the measures presented in this paper.

Another limitation is that only functional requirements were taken into consideration when developing the case study, hence it is not clear what impact non-functional requirements, such as security and performance, could have on the structure of software and business process models. Finally, statistical analyses were not conducted, and only a small subset of software engineering metrics was used in the case study.

5 Conclusions and Future Work

This paper has demonstrated the impact of Service-Oriented and Object-Oriented development paradigms on the structural software attributes of size, complexity, coupling, and cohesion using a case study developed with two contrasting approaches representing extremes of service granularity. The resulting designs and implementations were measured in three stages: after the initial system was developed; after the business logic for the initial system was changed; and after a new functionality was added to the initial system.

The quantitative comparison based on a set of eight mature software engineering metrics suggested that: i) systems developed using the SO approach could exhibit lower coupling between software modules compare to the ones developed with the OO approach; ii) the SO approach may provide better separation between business logic and software thus requiring less modifications upon the changes to the business logic; iii) however, systems developed using the OO approach has a potentially lower structural complexity. Based on the above suggestions, a conclusion can be made that there is a need for a balance between the presented extremes in service granularity in order to maintain low coupling and complexity while allowing for easier changes to business logic and rules.

To formalise the findings presented in this paper, a set of SOA-specific metrics for measuring structural software attributes will be identified in future work since it was discovered that OO-based metrics for measuring cohesion and complexity are not readily applicable to SOA. Such metrics will be applied to the data collected from a larger case study, facilitating more formal empirical analysis and validation of described hypotheses. In addition, the issues discussed in the paper should facilitate future research into the area of service granularity and implementation of business logic in software with the aim of specifying detailed guidelines and activities applicable to SOA development.

Acknowledgement. This project is funded by the ARC (Australian Research Council), under Linkage scheme no. LP0455234.

References

[1] Andrews, T., et al., Business Process Execution Language for Web Services, Version 1.1. 2003, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[2] Arsanjani, A., Service-oriented modeling and architecture: how to identify, specify, and realize services for your SOA. 2004, IBM - whitepaper. ftp://www6.software.ibm.com/software/developer/library/ws-soa-design1.pdf

[3] Briand, L.C., et al. A Comprehensive Empirical Validation of Design Measures for Object- Oriented Systems. in Fifth International Software Metrics Symposium. 1998.

[4] Chidamber, S.R. and C.F. Kemerer, A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 1994. 20(6): p. 476-493.

[5] Clune, J., BPEL in Service-Oriented Architecture, in Web Services Journal. 2005. p. 16-19.

[6] Dolado, J., A validation of the component-based method for software size estimation. IEEE Transactions on Software Engineering, 2000. 26(10): p. 1006-1021.

[7] Erl, T., Service-Oriented Architecture: a field guide to integrating XML and Web services. 2004, Upper Saddle River, NJ: Prentice Hall PTR.

[8] Fenton, N.E. and M. Neil, Software Metrics: Roadmap, in Future of Software Engineering, A. Finkelstein, Editor. 2000, ACM Press.

[9] Fowler, M., D. Rice, and D. Hoang, Patterns of enterprise application architecture. The Addison-Wesley signature series. 2003, Boston, Mass: Addison-Wesley. 533.

[10] Henderson-Sellers, B., Object-Oriented Metrics: Measures of Complexity. 1996, New Jersey, USA: Prentice-Hall.

[11] Jacobson, I., G. Booch, and J. Rumbaugh, The unified software development process. The Addison-Wesley object technology series. 1999, Reading, Mass: Addison-Wesley. 463.

[12] Kruchten, P., The Rational Unified Process: an introduction. 3 ed. 2003, Boston, MA: Addison-Wesley.

[13] McCabe, T.J. and A.H. Watson, Software Complexity. Journal of Defense Software Engineering, 1994. 7(12): p. 5-9.

[14] Pasley, J., How BPEL and SOA are changing Web services development. Internet Computing, IEEE, 2005. 9(3): p. 60-67.

[15] Perepletchikov, M., C. Ryan, and Z. Tari. The Impact of Software Development Strategies on Project and Structural Software Attributes in SOA. in Second INTEROP Network of Excellence Dissemination Workshop (INTEROP'05). 2005. Ayia Napa, Cyprus.

[16] Prnjat, O. and L. Sacks. Measuring complexity of network and service management components. in Second IEEE Latin American Network Operations and Management Symposium (LANOMS 2001). 2001. Belo Horizonte, Brazil.

[17] Singh, M.P. and M.N. Huhns, Service-Oriented Computing: Semantics, Processes, Agents. 2005, West Sussex, England: John Wiley & Sons.