How to Measure the Control-flow Complexity of Web Processes and Workflows

Jorge Cardoso, Department of Mathematics and Engineering, University of Madeira, Portugal

SUMMARY

Several Web process and workflow specification languages and systems have been developed to ease the task of modeling and supporting business processes. In a competitive e-commerce and e-business market, organizations want Web processes and workflows to be simple, modular, easy to understand, easy to maintain and easy to re-engineer.

To achieve these objectives, one can calculate the complexity of processes. The complexity of processes is intuitively connected to effects such as readability, understandability, effort, testability, reliability and maintainability. While these characteristics are fundamental in the context of processes, no methods exist that quantitatively evaluate the complexity of processes.

The major goal of this chapter is to describe a measurement to analyze the control-flow complexity of Web processes and workflows. The measurement is to be used at design-time to evaluate the complexity of a process design before implementation.

INTRODUCTION

The emergence of e-commerce has changed the foundations of business, forcing managers to rethink their strategies. Organizations are increasingly faced with the challenge of managing e-business systems, Web services, Web processes, and workflows.

Web Services and Web processes promise to ease several current infrastructure challenges, such as data, application, and process integration. With the emergence of Web services, a workflow management system become essential to support, manage, and enact Web processes, both between enterprises and within the enterprise (Sheth, Aalst, & Arpinar, 1999).

The effective management of any process requires modeling, measurement, and quantification. Process measurement is concerned with deriving a numeric value for attributes of processes. Measures, such as Quality of Service measures (Cardoso, Miller, Sheth, Arnold, & Kochut, 2004), can be used to improve processes productivity and quality.

To achieve an effective management, one fundamental area of research that needs to be explored is the complexity analysis of processes. Process complexity can be viewed as a component of a QoS model for processes, since complex processes are more prone to errors. For example, in software

engineering it has been found that program modules with high complexity indices have a higher frequency of failures (Lanning & Khoshgoftaar, 1994).

Surprisingly, in spite of the fact that there is a vast literature on software measurement of complexity, Zuse (Zuse, 1997) has found hundreds of different software metrics proposed and described, while no research on process complexity measurement has yet been carried out.

A Web process is composed of a set of Web services put together to achieve a final goal. As the complexity of a process increases, it can lead to poor quality and be difficult to reengineer. High complexity in a process may result in limited understandability and more errors, defects, and exceptions leading processes to need more time to develop, test and maintain. Therefore, excessive complexity should be avoided. For instance, critical processes, in which failure can result in the loss of human life, requires a unique approach to development, implementation and management. For this type of processes, typically found in healthcare applications (Anyanwu, Sheth, Cardoso, Miller, & Kochut, 2003), the consequences of failure are terrible. The ability to produce processes of higher quality and less complexity is a matter of endurance.

Our work borrows some techniques from the branch of software engineering known as software metrics, namely McCabe's cyclomatic complexity (MCC) (McCabe, 1976). A judicious adaptation and usage of this metric during development and maintenance of Web process applications can result in a better quality and maintainability. Based on MCC, we propose a control-flow complexity metric to be used during the design of processes. Web process control-flow complexity is a design-time metric. It can be used to evaluate the difficulty of producing a Web process before its implementation. When control-flow complexity analysis becomes part of the process development cycle, it has a considerable influence in the design phase of development, leading to further optimized processes. This control-flow complexity analysis can also be used in deciding whether to maintain or redesign a process.

Throughout this chapter, we will use the term "process" to refer to a Web process or a workflow and we will use the term "activity" to refer to a Web service or a workflow task.

CHAPTER STRUTURE

This chapter is structured as follows. The first section presents the related work. We will see that while a significant amount of work in the software engineering field has been developed to quantify the complexity of programs, the literature and work on complexity analysis for Web processes and workflow are inexistent. In the next section, we discuss the analysis of processes' complexity. We start by giving a definition for Web processes' complexity. We then enumerate a set of properties that are highly desirable for a model and theory to calculate the complexity of processes. In this section, we also motivate the reader towards a greater understanding of the importance and use of complexity metrics for processes. The next section

gives an overview of McCabe's cyclomatic complexity. This overview is important since our approach borrows some of McCabe's ideas to evaluate complexity. Subsequently, we discuss process control-flow complexity. We initiate this section giving the semantics of processes' structure and representation. Once the main elements of a process are identified and understood, we show how control-flow complexity can be calculated for processes. Finally, the last section presents our conclusions and future work.

RELATED WORK

While a significant amount of research on the complexity of software programs has been done in the area of software engineering, the work found in the literature on complexity analysis for Web processes and workflows is inexistent. Since the research on process complexity is inexistent, in this section we will discuss the progress made in the area of software complexity.

The last 30 years has seen a large amount of research aimed at determining measurable properties to capture the notions of complexity of software. The earliest measures were based on analysis of software code, the most fundamental being a basic count of the number of Lines of Code (LOC). Despite being widely criticized as a measure of complexity, it continues to have widespread popularity mainly due to its simplicity (Azuma & Mole, 1994).

An early measure, proposed by McCabe (McCabe, 1976), viewed program complexity related to the number of control paths through a program module. This measure provides a single number that can be compared to the complexity of other programs. It is also one of the more widely accepted software metrics. It is intended to be independent of language and language format.

The search for theoretically based software measures with predictive capability was pioneered by Halstead (Halstead, 1977). Complexity measurement was developed to measure a program module's complexity directly from source code, with emphasis on computational complexity. The measures were developed as a means of determining a quantitative measure of complexity based on a program comprehension as a function of program operands (variables and constants) and operators (arithmetic operators and keywords which alter program control flow).

Henry and Kafura (Henry & Kafura, 1981) proposed a metric based on the impact of the information flow in a program' structure. The technique suggests identifying the number of calls to a module (i.e. the flows of local information entering: fan-in) and identifying the number of calls from a module (i.e. the flows of local information leaving: fan-out). The measure is sensitive to the decomposition of the program into procedures and functions, on the size and the flow of information into procedures and out of procedures.

A recent area of research involving Web processes, workflows, and Quality of Service can also be considered related to the work in this chapter. Organizations operating in modern markets, such as e-commerce activities and distributed Web services interactions, require QoS management. Appropriate quality control leads to the creation of quality products and services; these, in turn, fulfill customer expectations and achieve customer satisfaction. Quality of service can be characterized according to various dimensions. For example, Cardoso et al. (Cardoso, Sheth, & Miller, 2002) have constructed a QoS model for processes composed of three dimensions: time, cost, and reliability. Another dimension that could be considered relevant under the QoS umbrella is the complexity of processes. Therefore, the complexity dimension could be added and integrated to the QoS model already developed (Cardoso, Miller et al., 2004).

PROCESS COMPLEXITY ANALYSIS

The overall goal of process complexity analysis is to improve the comprehensibility of processes. The graphical representation of most process specification languages provides the user with the capability to recognize complex areas of processes. Thus, it is important to develop methods and measurements to automatically identify complex processes and complex areas of processes. Afterwards, these processes can be reengineered to reduce the complexity of related activities. One key to the reengineering is the availability of a metric that characterizes complexity and provides guidance for restructuring processes.

Definition of Process Complexity

Several definitions have been given to describe the meaning of software complexity. For example, Curtis (Curtis, 1980) states that complexity is a characteristic of the software interface which influences the resources another system will expend or commit while interacting with the software. Card and Agresti (Card & Agresti, 1988) define relative system complexity as the sum of structural complexity and data complexity divided by the number of modules changed. Fenton (Fenton, 1991) defines complexity as the amount of resources required for a problem's solution.

After analyzing the characteristics and specific aspects of Web processes and workflows, we believe that the definition that is better suited to describe processes complexity can be derived from (IEEE, 1992). Therefore, we define process complexity as *the degree to which a process is difficult to analyze, understand or explain. It may be characterized by the number and intricacy of activity interfaces, transitions, conditional and parallel branches, the existence of loops, roles, activity categories, the types of data structures, and other process characteristics*.

Process Complexity Measurement Requirements

The development of a model and theory to calculate the complexity associated with a Web process or workflow need to conform to a set of basic but important properties. The metric should be easy to learn, computable,

consistent and objective. Additionally, the following properties are also highly desirable (Tsai, Lopex, Rodriguez, & Volovik., 1986; Zuse, 1990):

- Simplicity. The metric should be easily understood by its end users, i.e., process analysts and designers.
- Consistency. The metric should always yield the same value when two independent users apply the measurement to the same process, i.e. they should arrive at the same result.
- Automation. It must be possible to automate the measurement of processes.
- Measures must be additive. If two independent structures are put into sequence then the total complexity of the combined structures is at least the sum of the complexities of the independent structures.
- Measures must be interoperable. Due to the large number of existing specification languages, both in academia and industry, the measurements should be independent of the process specification language. A particular complexity value should mean the same thing whether it was calculated from a process written in BPEL (BPEL4WS, 2002), WSFL (Leymann, 2001), BPML (BPML, 2004), YAWL (Aalst & Hofstede, 2003), or some other specification language. The objective is to be able to set complexity standards and interpret the resultant numbers uniformly across specification languages.

These properties will be taken into account in the next sections when we introduce our model to compute the complexity of processes.

Uses of Complexity

Analyzing the complexity at all stages of process design and development helps avoid the drawbacks associated with high complexity processes. Currently, organizations have not implemented complexity limits as part of their business process management projects. As a result, it may happen that simple processes come to be designed in a complex way. For example, important questions that can be made relative to the process illustrated in Figure 2 (Anyanwu et al., 2003) are: “can the Eligibility Referral workflow be designed in a simpler way?”, “what is the complexity of the workflow?” and “what areas or regions of the workflow are more complex and therefore more prone to errors?”

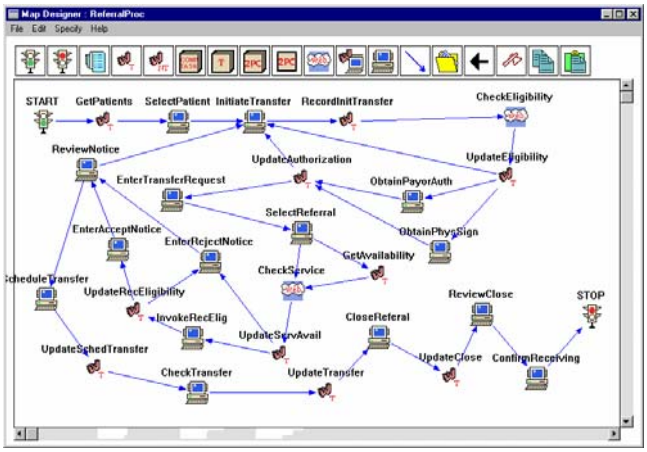

Figure 1. Eligibility Referral Workflow

The use of complexity analysis will aid in constructing and deploying Web processes and workflows that are more simple, reliable and robust. The following benefits can be obtained from the use of complexity analysis:

- Quality assessment. Processes quality is most effectively measured by objective and quantifiable metrics. Complexity analysis allows calculating insightful metrics and thereby identifying complex and error prone processes.
- Maintenance analysis. The complexity of processes tends to increase as they are maintained and over a period of time (Figure 2). By measuring the complexity before and after a proposed change, we can minimize the risk of the change.
- Reengineering. Complexity analysis provides knowledge of the structure of processes. Reengineering can benefit from the proper application of complexity analysis by reducing the complexity of processes.
- Dynamic behavior. Processes are not static applications. They are constantly undergoing revision, adaptation, change, and modification to meet end users needs. The complexity of these processes and their continuous evolution makes it very difficult to assure their stability and reliability. In-depth analysis is required for fixing defects in portions of processes of high complexity (Figure 2).

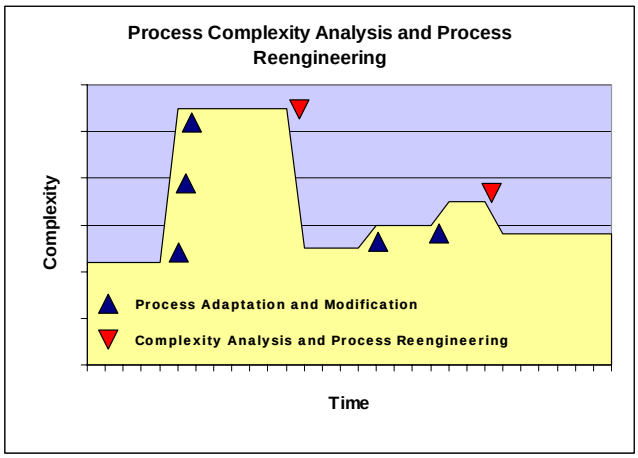

Figure 2. Process Complexity Analysis and Process Reengineering

OVERVIEW OF MCCABE'S CYCLOMATIC COMPLEXITY

Since our work to evaluate processes' complexity borrows some ideas from McCabe's cyclomatic complexity (MCC) (McCabe, 1976) to analyze software complexity, we start by describing the importance of MCC and illustrates its usage. This metric was chosen for its reliability as a complexity indicator and its suitability for our research.

Since its development, McCabe's cyclomatic complexity has been one of the most widely accepted software metrics and has been applied to tens of millions of lines of code in both the Department of Defense (DoD) and commercial applications. The resulting base of empirical knowledge has allowed software developers to calibrate measurements of their own software and arrive at some understanding of its complexity.

Software metrics are often used to give a quantitative indication of a program's complexity. However, it is not to be confused with algorithmic complexity measures (e.g. Big-Oh "O"-Notation), whose aim is to compare the performance of algorithms. Software metrics have been found to be useful in reducing software maintenance costs by assigning a numeric value to reflect the ease or difficulty with which a program module may be understood.

McCabe's cyclomatic complexity is a measure of the number of linearly independent paths in a program. It is intended to be independent of language and language format (McCabe & Watson, 1994). MCC is an indication of a program module's control flow complexity. Derived from a module's control graph representation, MCC has been found to be a reliable indicator of complexity in large software projects (Ward, 1989). This metric is based on the assumption that a program's complexity is related to the number of control paths through the program. For example, a 10-line program with 10 assignment statements is easier to understand than a 10-line program with 10 if-then statements.

MCC is defined for each module to be e - n + 2, where e and n are the number of edges and nodes in the control flow graph, respectively. Control flow graphs describe the logic structure of software modules. The nodes represent computational statements or expressions, and the edges represent transfer of control between nodes. Each possible execution path of a software module has a corresponding path from the entry to the exit node of the module's control flow graph. For example, in Figure 3, the MCC of the control flow graph for the Java code described is 14-11+2=5.

Node	Statement
(1)	while(x<100){
(2)	if (a[x] % 2 == 0) {
(3)	parity = 0;
	}
	else {
(4)	parity = 1;
(5)	}
(6)	switch(parity){
	case 0:
(7)	println("a[" + i + "] is even");
	case 1:
(8)	println("a[" + i + "] is odd");
	default:
(9)	println("Unexpected error");
	}
(10)	x++;
	}
(11)	p = true;

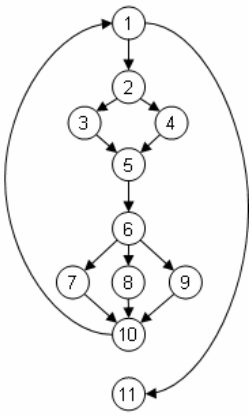

Figure 3. Example of a Java program and its corresponding flowgraph

Our major objective is to develop a metric that could be used in the same way as the MCC metric but to evaluate processes' complexity. One of the first important observations that can be made from MCC control flow graph, shown in Figure 3, is that this graph is extremely similar to Web processes and workflows. One major difference is that the nodes of a MCC control flow graph have identical semantics, while process nodes (i.e., Web services or workflow tasks) can have different semantics (e.g., AND-splits, XOR-splits, OR-joins, etc). Our approach will tackle this major difference.

PROCESS CONTROL-FLOW COMPLEXITY

Complexity metrics provide valuable information concerning the status and quality of process development projects. Access to this information is vital for accurately assessing overall process quality, identifying areas that need improvement, and focusing on development and testing efforts. In this section, we describe the structure and representation of Web processes and discuss how control-flow complexity is defined and computed for a Web process.

PROCESS STRUCTURE AND REPRESENTATION

Control flow graphs can be used to describe the logic structure of Web processes. A Web process is composed of Web services and transitions. Web services are represented using circles and transitions are represented using arrows. Transitions express dependencies between Web services. A Web service with more than one outgoing transition can be classified as an AND-split, OR-split or XOR-split. AND-split Web services enable all their outgoing transitions after completing their execution. OR-split Web services enable one or more outgoing transition after completing their execution. XOR-split Web services enable only one outgoing transition after completing their execution. AND-split Web services are represented with a '•', OR-split are represented with a 'O' and XOR-split Web services are represented with a '⊕'. A Web service with more than one incoming transition can be classified as an AND-join, OR-join or XOR-join. AND-join Web services start their execution when all their incoming transitions are enabled. OR-join services start their execution when a subset of their incoming transitions is enabled. XOR-join Web services are executed as soon as one of the incoming transitions is enabled. As with AND-split, OR-split and XOR-split Web services, AND-join, OR-join and XOR-join Web services are represented with the symbols '•', 'O' and '⊕', respectively.

An example of a Web process is shown in Figure 4. The process has been developed by the Fungal Genome Resource (FGR) laboratory in an effort to improve the efficiency of their processes (Cardoso, Miller et al., 2004). One of the reengineered processes was the DNA sequencing workflow, since it was considered to be beneficial for the laboratory's daily activities.

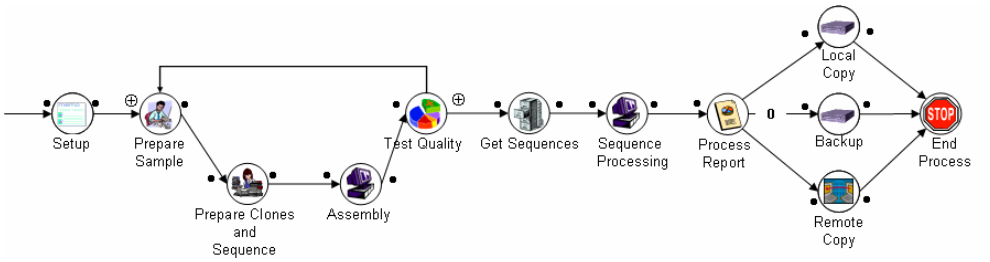

Figure 4. The DNA Sequencing Workflow.

Semantics of Processes

The complexity of a Web process or workflow can be analyzed according to different perspectives. In our work we are interested in evaluating the complexity of processes from a control-flow perspective. In a Web process and workflow the control-flow logic is captured in a process model and function logic is captured in the applications, data, and people the model invokes. A process model includes basic constructs such as transitions, roles, Web services or tasks, XOR-splits, OR-splits, AND-splits, XOR-joins, OR-joins, AND-joins and networks (sub-processes.)

Our approach uses the idea introduced by McCabe. Numerous studies and experience in software projects have shown that the MCC measure correlates very closely with errors in software modules. The more complex a module is, the more likely it is to contain errors. Our goal is to adapt McCabe's cyclomatic complexity to be applied to processes.

As stated previously, one interesting remark is that all the nodes of MCC flowgraphs have identical semantics. Each node represents one statement in a source code program. On the other hand, the nodes in Web processes and workflows can assume different semantics. Thus, we consider three constructs with distinct semantics presents in process models: XOR-split, OR- split, and AND-split. The three constructs have the following semantics:

- XOR-split. A point in the process where, based on a decision or process control data, one of several transitions is chosen. It is assumed that only one of the alternatives is selected and executed, i.e. it corresponds to a logic exclusive OR.
- OR-split. A point in the process where, based on a decision or process control data, one or more transitions are chosen. Multiple alternatives are chosen from a given set of alternatives. It is assumed that one or more of the alternatives is selected and executed, i.e. it corresponds to a logic OR.
- AND-split. This construct is required when two or more activities are needed to be executed in parallel. During the execution of a process, when an AND-split is reached the single thread of control splits into multiple treads of control which are executed in parallel, thus allowing activities to be executed at the same time or in any order. It is assumed that all the alternatives are selected and executed, i.e. it corresponds to a logic AND.

Definition and Measurement of Control-flow Complexity

The control-flow behavior of a process is affected by constructs such as splits and joins. Splits allow defining the possible control paths that exist through the process. Joins have a different role; they express the type of synchronization that should be made at a specific point in the process.

Since we are interested in calculating the complexity of processes' control-flow, the formulae that we will present evaluate the complexity of XOR-split, OR-split, and AND-split constructs. We call this measurement of complexity, Control-flow Complexity (CFC). Each formula computes the number of states that can be reached from one of the three split constructs. The measure is based on the relationships between mental discriminations needed to understand a split construct and its effects. This type of complexity has been referred to as psychological complexity. Therefore, the more possible states follow a split, the more difficulty the designer or business process engineer has to understand the section of a processes and thus the process itself.

In processes, the McCabe's Cyclomatic complexity cannot be used successfully since the metric ignores the semantics associated with nodes of the graph. While the nodes (i.e. activities) of processes have distinct semantics associated, the nodes of a program's flowgraph are undifferentiated.

We now introduce several definitions that will constitute the basis for CFC measurement.

Definition 1 (Process measurement). Process measurement is concerned with deriving a numeric value for an attribute of a process.

Examples of attributes can include process complexity, duration (time), cost, and reliability (Cardoso, Miller et al., 2004).

Definition 2 (Process metric). Any type of measurement related to a process. Process metrics allows attributes of processes to be quantified.

Definition 3 (Activity fan-out). Fan-out is the number of transitions going out of an activity. The fan-out is computed using function *fan-out(a)*, where *a* is n activity.

Definition 4 (Control-flow induced mental state). A mental state is a state that has to be considered when a designer is developing a process. Splits introduce the notion of mental states in processes. When a split (XOR, OR, or AND) is introduced in a process, the business process designer has to mentally create a map or structure that accounts for the number of states that can be reached from the split.

The notion of mental state is important since there are certain theories (Miller, 1956) that prove complexity beyond a certain point defeats the human mind's ability to perform accurate symbolic manipulations, and hence results in error.

Definition 5 (XOR-split Control-flow Complexity). XOR-split control-flow complexity is determined by the number of mental states that are introduced with this type of split. The function $CFC_{XOR\text{-}split}(a)$, where *a* is a activity, computes the control-flow complexity of the XOR-split *a*. For XOR-splits, the control-flow complexity is simply the fan-out of the split.

$$CFC_{XOR\text{-}split}(a) = \text{fan-out}(a)$$

In this particular case, the complexity is directly proportional to the number of activities that follow a XOR-split and that a process designer needs to consider, analyze, and assimilate. The idea is to associate the complexity of an XOR-split with the number of states (Web services or

workflow tasks) that follow the split. This rationale is illustrated in Figure 5. Please note that in this first case the computation and result bear a strong similarity to the McCabe's cyclomatic complexity.

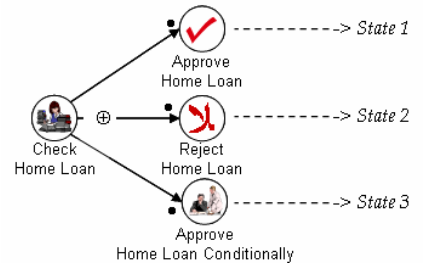

Figure 5. XOR-split control-flow complexity

Definition 6 (OR-split Control-flow Complexity). OR-split control-flow complexity is also determined by the number of mental states that are introduced with the split. For OR-splits, the control-flow complexity is 2^n-1, where n is the fan-out of the split.

$$CFC_{OR\text{-}split}(a) = 2^{fan\text{-}out(a)} - 1$$

This means that when a designer is constructing a process he needs to consider and analyze 2^n-1 states that may arise from the execution of an OR-split construct.

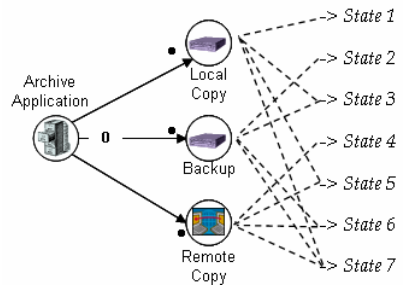

Figure 6. OR-split control-flow complexity

Mathematically, it would appear more obvious that 2^n states can be reached after the execution of an OR-split. But since a process that has started its execution has to finish, it cannot be the case where after the execution of an OR-split no transition is activated, i.e. no Web service or workflow task is executed. Therefore, this situation or state cannot happen.

Definition 7 (AND-split Control-flow Complexity). For an AND-split, the complexity is simply 1.

$$CFC_{AND\text{-}split}(a) = 1$$

The designer constructing a process needs only to consider and analyze one state that may arise from the execution of an AND-split construct since it is assumed that all the outgoing transitions are selected and followed.

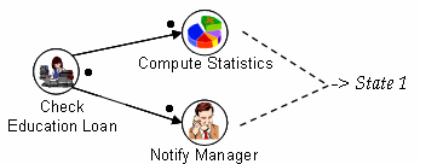

Figure 7. AND-split control-flow complexity

The higher the value of $CFC_{XOR\text{-}split}(a)$, $CFC_{OR\text{-}split}(a)$, and $CFC_{AND\text{-}split}(a)$, the more complex is a process's design, since developers have to handle all the states between control-flow constructs (splits) and their associated outgoing transitions and activities. Each formula to calculate the complexity of a split construct is based on the number of states that follow the construct.

CONTROL-FLOW COMPLEXITY OF PROCESSES

Mathematically, control-flow complexity metric is additive. Thus, it is very easy to calculate the complexity of a process, by simply adding the CFC of all split constructs. The control-flow complexity is calculated as follows, where *p* is a Web process or workflow.

$$CFC(p) = \sum_{ws \in \{xor-splits \in p\}} CFC_{XOR-split}(ws) + \sum_{ws \in \{or-splits \in p\}} CFC_{OR-split}(ws) + \sum_{ws \in \{and-splits \in p\}} CFC_{AND-split}(ws)$$

The greater the value of the CFC, the greater the overall architectural complexity of a process. CFC analysis seeks to evaluate complexity without direct execution of processes.

Example of CFC Calculation

As an example, let us take the Web process shown in Figure 8 and calculate its CFC. The process has been developed by a bank that has adopted a workflow management system (WfMS) to support its business processes. Since the bank supplies several services to its customers, the adoption of a WfMS has enabled the logic of bank processes to be captured in Web processes schema. As a result, all the services available to customers are stored and executed under the supervision of the workflow system. One of the services supplied by the bank is the loan application

process depicted in Figure 8. The Web process is composed 21 Web services, 29 transitions, three XOR-splits (Check Loan Type, Check Home Loan, Check Car Loan), one OR-split (Archive Application) and one AND-split (Check Education Loan).

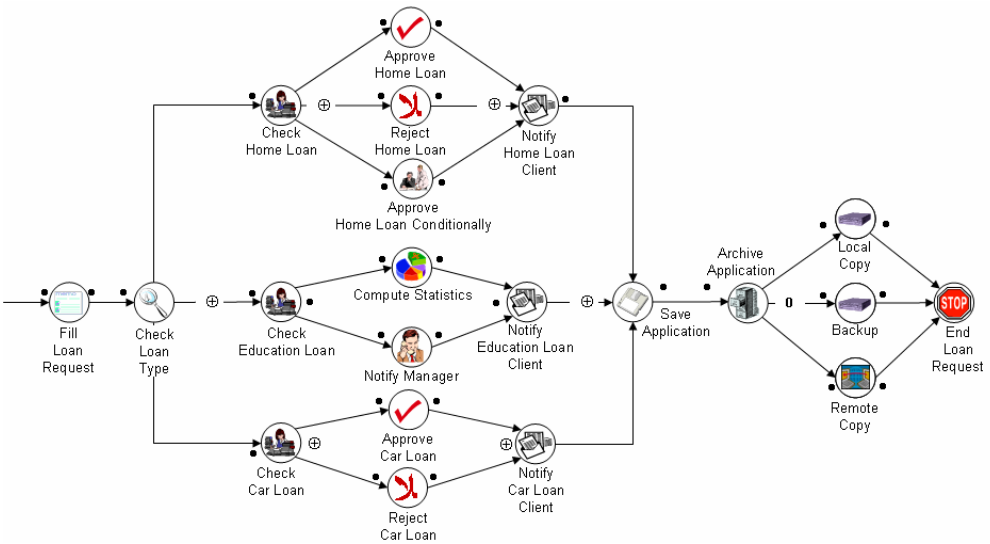

Figure 8. The Loan Application Process

It was decided that before placing the Web process in a production environment, a process complexity analysis was required to evaluate the risk involved with the reengineering effort. The results of the control-flow complexity analysis carried out are shown in
Table 1.

Table 1. CFC metrics for the Web process from Figure 8

Split	CFC
$CFC_{XOR\text{-}split}$(Check Loan Type)	3
$CFC_{XOR\text{-}split}$(Check Home Loan)	3
$CFC_{XOR\text{-}split}$(Check Car Loan)	2
$CFC_{OR\text{-}split}$(Archive Application)	2^3-1
$CFC_{AND\text{-}split}$(Check Education Loan)	1
CFC(Loan Application)	=16

From these values the control-flow complexity can be easily calculated. It is sufficient to mathematically add the CFC of each split. Thus, the resulting CFC value is 16 (i.e., $3+3+2+2^3-1+1$).

Since the results of the CFC analysis gave a value considered to be low, it was determined that the Web process has a low complexity and therefore its implementation presented a low risk for the bank. Therefore, the Web process was deployed and implemented in a production environment. The CFC is a good indicator of the complexity of a process. As further research is conducted in this area it will become clear that in many cases it is necessary to limit CFC of Web process applications. Overly complex processes are more prone to errors and are harder to understand, test, and adapt.

One important question that needs to be investigated and answered is what is both the meaning of a given metric (for example, what is the significance of the CFC of 16 obtained in our example) and the precise number to use as a CFC limit in a process development. This answer will be given from empirical results only when organizations have successfully implemented complexity limits as part of their process development projects. For example, when using McCabe complexity metrics, the original limit of 10 indicates a simple program, without much risk, a complexity metric between 11 and 20 designates a more complex program with moderate risk, a metric between 21 and 50 denote a complex program with high risk. Finally, a complexity metric greater than 50 denotes an untestable program with a very high risk. We expect that limits for CFC will be obtained and set in the same way, using empirical and practical results from research and from real world implementation.

Verification

To test the validity of our metric, we have designed a small set of processes. A group of students has rated each process according to their perceived complexity. The students had previously received a 15-hour course on process design and implementation. We have then used our CFC measurement to calculate the complexity of each process design. Preliminary data analysis performed on the collected data led to some interesting results. A correlation was found between the perceived complexity and the control-flow complexity measure.

Based on these preliminarily interesting results, we are now starting a project that will have as an objective the development of a large set of empirical experiments involving process designs. The purpose is to find the degree of correlation between the perceived complexity that designers and business engineers have when studying and designing a process and the results obtained from applying our control-flow complexity measure.

CONCLUSIONS AND FUTURE WORK

Business Process Management Systems (BPMS)(Smith & Fingar, 2003) provide a fundamental infrastructure to define and manage business processes, Web processes and workflows. BPMS, such as Workflow

Management Systems (Cardoso, Bostrom, & Sheth, 2004) become a serious competitive factor for many organizations.

Our work presents an approach to carry out process complexity analysis. We have delineated the first steps towards using a complexity measurement to provide concrete Web process and workflow design guidance. The approach and the ideas introduced are worth exploring further since Web processes are becoming a reality in e-commerce and e-business activities.

In this chapter we propose a control-flow complexity measurement to be used during the design of processes. Process control-flow complexity is a design-time measurement. It can be used to evaluate the difficulty of producing a Web process design before implementation. When control-flow complexity analysis becomes part of the process development cycle, it has a considerable influence in the design phase of development, leading to further optimized processes. The control-flow complexity analysis can also be used in deciding whether to maintain or redesign a process. As known from software engineering, it is a fact that it is cost-effective to fix a defect earlier in the design lifecycle than later. To enable this, we introduce the first steps to carry out process complexity analysis.

Future directions of this work are to validate the complexity measurement to ensure that clear and confident conclusions can be drawn from its use. In addition to this, although the validity of the proposed complexity measurement was tested using a few empirical studies that formed the basis for its development, further work is required to validate its usability in contexts other than the ones in which the method was developed. In order to achieve these goals, it is necessary to evaluate a variety of processes and produce automated tools for measuring complexity features.

REFERENCES

Aalst, W. M. P. v. d., & Hofstede, A. H. M. t. (2003). *YAWL: Yet Another Workflow Language (Revised Version).* (QUT Technical report FIT-TR-2003-04). Brisbane: Queensland University of Technology2003.

Anyanwu, K., Sheth, A., Cardoso, J., Miller, J. A., & Kochut, K. J. (2003). Healthcare Enterprise Process Development and Integration. *Journal of Research and Practice in Information Technology, Special Issue in Health Knowledge Management, 35*(2), 83-98.

Azuma, M., & Mole, D. (1994). Software Management Practice and Metrics in the European Community and Japan: Some Results of a Survey. *Journal of Systems and Software, 26*(1), 5-18.

BPEL4WS. (2002). *Web Services*. IBM. Retrieved, from the World Wide Web: http://www-106.ibm.com/developerworks/webservices/

BPML. (2004). *Business Process Modeling Language*. Retrieved, 2004, from the World Wide Web: http://www.bpmi.org/

Card, D., & Agresti, W. (1988). Measuring Software Design Complexity. *Journal of Systems and Software, 8*, 185-197.

Cardoso, J., Bostrom, R. P., & Sheth, A. (2004). Workflow Management Systems and ERP Systems: Differences, Commonalities, and Applications. *Information Technology and Management Journal. Special issue on Workflow and E-Business (Kluwer Academic Publishers), 5*(3-4), 319-338.

Cardoso, J., Miller, J., Sheth, A., Arnold, J., & Kochut, K. (2004). Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web Journal, 1*(3), 281-308.

Cardoso, J., Sheth, A., & Miller, J. (2002). *Workflow Quality of Service.* Paper presented at the International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference (ICEIMT/IEMC'02), Valencia, Spain.

Curtis, B. (1980). Measurement and Experimentation in Software Engineering. *Proceedings of the IEEE, 68*(9), 1144-1157.

Fenton, N. (1991). *Software Metrics: A Rigorous Approach.* London: Chapman & Hall.

Halstead, M. H. (1977). Elements of Software Science, Operating, and Programming Systems Series (Vol. 7). New York, NY: Elsevier.

Henry, S., & Kafura, D. (1981). Software Structure Metrics Based On Information-Flow. *IEEE Transactions On Software Engineering, 7*(5), 510-518.

IEEE. (1992). *IEEE 610, Standard Glossary of Software Engineering Terminology.* New York: Institute of Electrical and Electronic Engineers.

Lanning, D. L., & Khoshgoftaar, T. M. (1994). Modeling the Relationship Between Source Code Complexity and Maintenance Difficulty. *Computer, 27*(9), 35-41.

Leymann, F. (2001). *Web Services Flow Language (WSFL 1.0).* IBM Corporation. Retrieved, from the World Wide Web: http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

McCabe, T. (1976). A Complexity Measure. *IEEE Transactions of Software Engineering, SE-2*(4), 308-320.

McCabe, T. J., & Watson, A. H. (1994). Software Complexity. *Crosstalk, Journal of Defense Software Engineering, 7*(12), 5-9.

Miller, G. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review.*

Sheth, A. P., Aalst, W. v. d., & Arpinar, I. B. (1999). Processes Driving the Networked Economy. *IEEE Concurrency, 7*(3), 18-31.

Smith, H., & Fingar, P. (2003). *Business Process Management (BPM): The Third Wave*: Meghan-Kiffer Press.

Tsai, W. T., Lopex, M. A., Rodriguez, V., & Volovik., D. (1986). *An approach measuring data structure complexity.* Paper presented at the COMPSAC 86.

Ward, W. (1989). Software Defect Prevention Using McCabe's Complexity Metric. *Hewlett Packard Journal, 40*(2), 64-69.

Zuse, H. (1990). *Software Complexity Measures and Models*. New York, NY: de Gruyter & Co.

Zuse, H. (1997). *A Framework of Software Measurement*. Berlin: Walter de Gruyter Inc.

INDEX