Quantifying the Complexity of IT Service Management Processes

Yixin Diao and Alexander Keller

IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
{diao, alexk}@us.ibm.com

Abstract. Enterprises and service providers are increasingly looking to process-based automation as a means of containing and even reducing the labor costs of systems management. However, it is often hard to quantify and predict the additional complexity introduced by IT service management processes before they actually have been deployed. Our approach consists in looking at this problem from a different, new perspective by regarding complexity as a surrogate for potential labor cost and human-error-induced problems: In order to effectively evaluate the benefits of IT service management processes – and to target the types of processes that contribute most to management complexity and cost – we need a set of metrics for quantifying the complexity and human cost of carrying out IT service management processes. This paper proposes such measures, and demonstrates how they can be applied to a typical service delivery process in order to assess its complexity hotspots as a basis for process re-engineering.

1 Introduction

The complexity of managing computing systems and information technology (IT) processes represents a major impediment to efficient, high-quality, error-free, and cost-effective service delivery, ranging from small-business servers to global-scale enterprise backbones. In order to accomplish these goals, enterprises and service providers turn increasingly to the IT Infrastructure Library (ITIL) [1]. ITIL comprises disciplines such as service management, support and delivery, and has established itself as the most widely used standardized process-based approach to IT service management. When implementing ITIL by means of IT management processes, however, one needs to be able to quantitatively measure the degree of IT management complexity exposed by particular processes, so that process designers and architects can discover complexity hotspots early in the design phase, and optimize the IT processes to reduce their complexity.

We regard complexity as a surrogate for potential labor cost and human-error-induced problems: IT systems and processes with a high degree of complexity demand humans and expertise to manage that complexity, increasing the total cost of ownership. Likewise, complexity increases the amount of time that must be spent interacting with a computing system or between administrators to perform the desired function, and therefore decreases efficiency and productivity.

R. State et al. (Eds.): DSOM 2006, LNCS 4269, pp. 61–73, 2006.

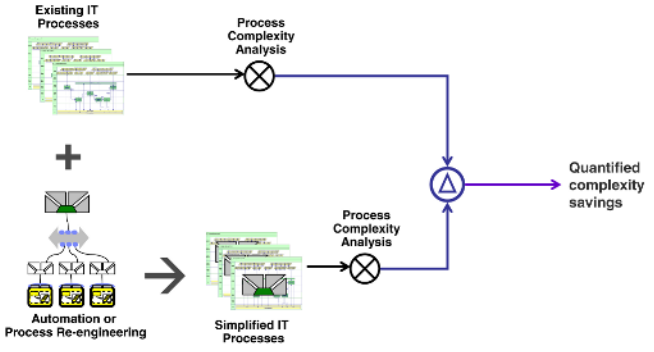

Fig. 1. Quantitative IT management process complexity evaluation

Furthermore, complexity results in human errors, as complexity challenges human reasoning and results in erroneous decisions even by skilled administrators. The goal is to reduce IT process complexity by designing, architecting, implementing, and assembling systems and processes with minimal complexity level.

There is little existing work in the area of process complexity analysis: Business process modeling tools typically include a process simulator, which allows the designer to run a set of simulated process executions (either through Monte Carlo or discrete event simulation) in order to assess the performance of the process, generate statistics about its execution, and pinpoint potential areas of improvement and optimization. However, many parameters need to be specified by the designer a priori: for example, for each task (or action), a duration is assigned during the modeling phase; for decisions, the designer needs to indicate up-front a percentage for each of the branches that indicates the probability that it will be taken (the sum of all branch probabilities equals 100%). None of the available business process model simulators focuses on process complexity as described in this paper. There is no overlap between existing work in the process simulation area and the work presented in this paper.

Related work in the system administration discipline has been carried out with a focus on establishing cost models, which take into account the impact of decisions. The most relevant work is [4], which generalizes an initial model for estimating the cost of downtime [7], based on the previously established System Administration Maturity Model [5]. Other related work can be found in information theory (e.g., Kolmogorov complexities [6]) or quality management in manufacturing (e.g., Six Sigma [9]).

Figure 1 illustrates our vision of quantitative IT management process complexity evaluation: Once a set of IT processes has been designed and architected, a process complexity analysis is carried out to pinpoint possible complexity hotspots and inefficiencies (depicted at the top of the Figure). In order to mitigate or even eliminate complexity hotspots, techniques like IT process re-engineering can be used to re-design the process(es); in addition, activities within a process that exhibit a very high degree of complexity are identified as candidates that should be further modeled and investigated. For example, they

can be broken down to human comprehensible sub-tasks, which can be further supported by tooling or delegated to automation. In a third step, the process complexity analysis is applied to the simplified IT processes (depicted at the bottom of the figure). Finally, the results of the 'before' and 'after' analyses are evaluated side-by-side to obtain quantified complexity savings in order to measure the improvements.

The focus of this paper is the process complexity analysis because the process complexity model and its measures are the basis of both 'before' and 'after' analyses of the overall quantitative process complexity evaluation. The paper is organized as follows: Section 2 overviews our model for IT management processes. Section 3 details the complexity measures for quantifying processes and Section 4 describes the tooling we have implemented. In Section 5, we perform a process complexity analysis by applying our model. Our conclusions are contained in section 6.

2 IT Process Complexity Model

This section describes the proposed model of IT process complexity, which is used for computing the operational complexity of IT processes. Configuration management, change management, release management, and problem management are all examples of IT service management processes.

In [2], we have introduced a model for assessing configuration complexity. The intent of that model was to capture and quantify the complexity of a straight-line flow through a configuration procedure. Three different complexity measures have been identified, which are briefly summarized below in order to provide some background for the following discussion: *Execution Complexity* refers to the complexity involved in performing the configuration actions that make up the configuration procedure, typically characterized by the number of actions and the context switch distances between actions. *Parameter Complexity* is the complexity involved in providing configuration data to the computer system during a configuration procedure. *Memory Complexity* takes into account the number of parameters that must be remembered, the length of time they must be retained in memory, and how many intervening items were stored in memory between uses of a remembered parameter.

While the above three complexity measures are sufficient to capture the interactions of an administrator with a managed system at runtime, we need to extend this model to provide a quantitative assessment of IT management complexity that involves:

1. interaction between 2 or more roles in a process,
2. passing data in various formats (a.k.a., business items) between tasks, and
3. decision making among multiple roles.

Specifically, we model a process as a set of roles, each of which may participate in a set of tasks that either consume or produce business items. As depicted in Figure 2, IT management processes are modeled using the following three components: Roles, Tasks, and Business Items. This is consistent with

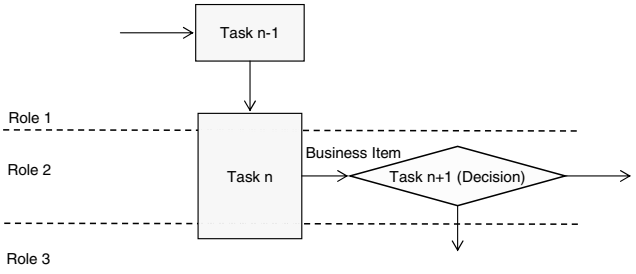

Fig. 2. Complexity model extensions for IT management processes

how common-off-the-shelf business process modeling tools, such as IBM WebSphere Business Modeler (WBM) or Tibco Business Studio structure business processes. Consequently, the information needed for complexity calculation can be extracted from typical process model data (see also section 4).

3 IT Process Complexity Measures

This section describes the per-task complexity metrics for each of the three aspects of the process complexity. Note that these metrics are designed to capture the first-order effects. However, our experience shows that even at this level they have demonstrated the effectiveness in identifying key automation opportunities and complexity bottlenecks, and in tracking process improvement.

3.1 Execution Complexity

Execution complexity covers the complexity involved in performing the tasks that make up the IT process. We use two metrics for execution complexity: base execution complexity and decision complexity.

Base Execution Complexity indicates complexity of the task according to its execution type. Values for this score are assigned according to a weighting scale of different task types. Each role involved in the task must be assigned an execution type chosen from below with corresponding values shown in square brackets next to the type name. The base execution complexity for that task is then the sum of values from all the roles. That is, for a task involving R roles ($r = 1, 2, \ldots, R$), its base execution complexity is computed as

$$E_{base} = \sum_{r=1}^{R} execType(r) \tag{1}$$

where the execution type $execType(r)$ is defined with regard to three types. *automatic* [0] - if the task is fully automated. *toolAssisted* [1] - if the task is manual, but tool-assisted. For example, manual triggering of a provisioning workflow or script involves providing workflow definitions. A service invocation

task is also classified as toolAssisted, as it transfers the work to an external role. *manual* [2] - if the task is a fully manual procedure.

The above approach defines a normalized, unit-less score for base execution complexity. However, there are two possible extensions: (1) If data on average task times is available, they can be used to define the base execution complexity. (2) If task level procedure complexity analysis has been conducted, the base execution complexity can be defined with regard to procedure-wide execution, parameter, and memory complexity. Note that as a starting point we choose metric values in a linear scale (0, 1, 2) throughout the model; however, further studies need to be conducted to determine if a quadratic or exponential scale is more appropriate.

Decision Complexity quantifies additional execution complexity due to decision making. For non-decision making task, its value is zero. If a decision needs to be made, its complexity is based on the following four sub-metrics.

- *nBranches*: the number of branches in the decision. The intuition is that more choices result in higher decision complexity.
- *gFactor*: the degree of guidance. That is, how much information is provided to the user to make the correct choice. This is quantified with a three-level scale of increasing complexity: [0] for a specific (correct) recommendation of the decision branch to follow, [1] if general information is provided relating choices to goals so that a user could extract the correct decision with some processing of the information, and [2] if no information is provided to help the user select the correct choice.
- *cFactor*: the consequence of impact. That is, how significant is the impact if a wrong decision has been made. This is also quantified with a three-level scale of increasing complexity: [1] for negligible consequence, [2] for moderate consequence, and [3] for severe consequence.
- *vFactor*: the visibility of impact. That is, how much information is provided to the user to illustrate the consequences of their choice. This is also quantified with a three-level scale of increasing complexity: [1] for immediate consequence, [2] for short-term consequence, and [3] for long-term consequence in terms of end-state features.

With the above three sub-metrics, the decision complexity for that task is then multiplied by the number of roles (R) involved in that task.

$$E_{decision} = R \times (nBranches - 1) \times gFactor \times cFactor \times vFactor \qquad (2)$$

The above formula reflects the following intuition: the decision complexity is zero if the number of decision branches is one (no decision, straight flow); clear guidance reduces decision-making tasks to non-decision making tasks, even if it may involve multiple branches; immediate choice consequence may make the task tedious, if there are multiple branches and the guidance is unclear, but the level of complexity remains low since the decision uncertainty does not exist. Although the decision complexity formula may be expressed differently, we choose to use a multiplication format as we view the effects of multiple factors as being

orthogonal to each other. For example, if the guidance is clear ($gFactor = 0$), the decision complexity is 0 no matter how many branches it may have. This is the case where the system administrator needs to classify problem tickets to many different categories such as 'file system full' and 'high application response time'.

3.2 Coordination Complexity

The per-task metrics for coordination complexity are computed based on the roles involved and whether or not business items are transferred.

Coordination Link Complexity is indicated by a unit-less value representing the complexity of coordinating between multiple roles. For each link between the task under consideration and other tasks carried out by different roles, score values are assigned according to a weighting scale of different coordination link complexities. The coordination link complexity for that task is then the sum of values from all the links ($l = 1, 2, \ldots, L$) multiplied by the number of roles (R) involved in that task.

$$C_{link} = R \times \sum_{l=1}^{L} linkType(l) \tag{3}$$

where the link type $linkType(l)$ is defined as follows. $autoLink$ [0] - if it is linking to an automated task. $controlLink$ [1] - if it is a control flow link to a non-automated task without any business item being transferred. $dataTransferred$ [2] - if business items are transferred. $dataAdapted$ [3] - if the transferred business items need to be adapted. For example, adaptation is needed if the consuming role is automated and the source data format is not machine-readable.

We also define **Shared Task Complexity** for tasks that involve multiple roles. For example, conducting a change review meeting requires the participation of multiple roles. The shared task complexity is computed from the assigned score below multiplied by the number of roles (R) involved in that task.

$$C_{task} = R \times taskType \times (meetingIndicator + 1) \tag{4}$$

where $taskType$ is defined as follows. $notShared$ [0] - if it is not a shared task. $shared$ [1] - if it is a shared task. $BIConsumed$ [2] - if business items are consumed. The intuition is that coordination of data input requires extra cost. $BIProduced$ [3] - if business items are produced. The intuition is that coordination of data output is more expensive because it requires agreement among multiple roles. The meeting indicator $meetingIndicator$ takes a Boolean (0, 1) value: it takes a value of 0 if there is no meeting involved; otherwise, if a meeting is required, it takes a value of 1. Thus, having a meeting raises the shared task complexity by a factor of 2.

3.3 Business Item Complexity

The per-task business item (BI) complexity is computed based on the business items produced by the task under consideration.

Base BI Complexity is indicated by a unit-less value representing the complexity of involving business items. That is, for a task involving R roles and I business items either consumed or produced by that task, its base BI complexity is computed as

$$B_{base} = R \times I \tag{5}$$

BI source complexity is indicated by a unit-less value representing the complexity of supplying this field's value. Values for this score are assigned according to a weighting scale of different source complexities. Each field used in the business item must be assigned a source type chosen from one of the values given below. The field source complexity value for that field is then the value shown in square brackets below next to the type name. For each produced business item, a source complexity is assigned to each field based on the source that provides the field's data. Then, each source score is summed across the business items to produce the final per-task metric. That is, for a task involving R roles ($r = 1, 2, \ldots, R$), producing I_P business items ($i = 1, 2, \ldots, I_P$), and f fields ($f = 1, 2, \ldots, F_i$), its per-task business item complexity is computed as

$$B_{source} = R \times \sum_{i=1}^{I_P} \sum_{f=1}^{F_i} sourceScore(i, f) \tag{6}$$

where $sourceScore(i, f)$ is defined as follows: *internal* [0] - if the field value was produced from automation. *freeChoice* [1] - if the field value can be chosen freely, e.g., a new password. *documentationDirect* [2] - if the field value was taken directly from the task documentation, an online source, or a process description manual (e.g., a Redbook), without extrapolation or adaptation. *documentationAdapted* [3] - if the field value was extrapolated from an example in the task documentation, an online source, or a process description manual. *bestPractice* [4] - if the field value would be obvious to a system operator versed in the administrative best practices for the application domain. *environmentFixed* [5] - if the field value is constrained by the environment to a specific value that is selected by the operator after further research. *environmentConstrained* [6] - if the parameter value is constrained by the environment to a limited set of possible choices.

These seven sources are ranked in order of increasing complexity burden. For example, a field sourced internally or pulled straight from the documentation does not require the system administrator to figure out its value. On the other hand, a parameter constrained by the environment, where that constraint is not obvious, imposes significant complexity since the system administrator needs to infer the possible legal value.

4 Process Complexity Model Tooling

We will now discuss the architecture, design and implementation of the tooling that we have developed in order to support the process complexity model and

its measures that we have described in the previous sections. Our Integrated Complexity Analyzer is in charge of computing the complexity scores, according to the measures described in section 3.

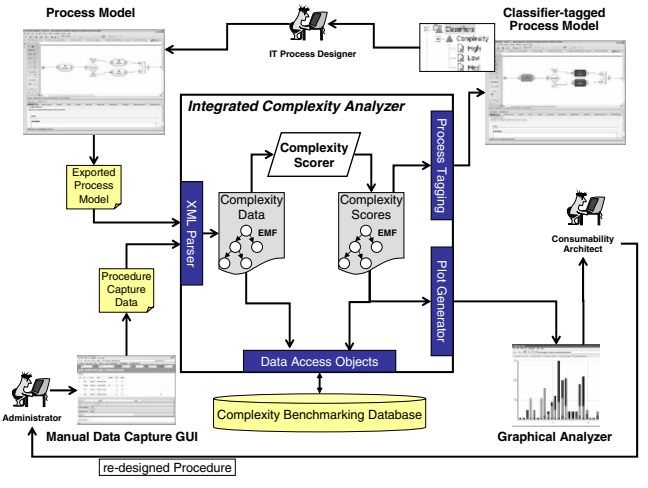

Fig. 3. Architecture of the Integrated Complexity Analyzer

4.1 Complexity Evaluation Scenarios

As depicted in Figure 3, there are two possible scenarios that need to be supported by the architecture of the Integrated Complexity Analyzer. The first scenario, depicted at the top of the figure, addresses complexity analysis at *process design time* and is the subject of this paper: An IT management process designer models a process by means of a common-off-the-shelf business process modeling tool, in our case IBM WebSphere Business Modeler version 6 [10]. In order to determine the complexity of the process and to pinpoint complexity hotspots, the designer exports the process to a file in the XML metadata interchange (XMI) format and executes our Integrated Complexity Analyzer. The latter automatically captures complexity data directly from the XMI file, computes the complexity scores and annotates the process model by tagging it with classifiers. The process designer can then re-import the process model and identifies, by means of color-coded activities that correspond to the different degrees of complexity, which activities and roles are considered complexity hotspots and should be simplified, if possible. Note that the redesign of the process model involves domain-specific knowledge and thus requires the involvement of the process designer. Once the model has been redesigned, the analysis is repeated until the complexity hotspots have been addressed.

The second scenario consists in having an administrator capture the configuration actions and parameters while executing a setup, change or configuration procedure for a product on a (distributed) system. This scenario, along with its

measures, is the subject of [2]. It is depicted at the bottom of Figure 3. The major differences to the first scenario are as follows: First, the complexity analysis is performed at *runtime* and therefore reflects a straight-line flow through the procedure. In addition, all complexity data is gathered manually (by means of a web based manual data capture GUI) as no tooling is available that would log the actions automatically; furthermore, assigning complexity ratings to configuration parameters requires the involvement of an administrator. Finally, the consumers and producers of the complexity data are typically different people: while the administrator is needed to perform the configuration procedure and capture the procedure complexity data, the consumer of the complexity scores is typically a so-called consumability architect, whose role consists in diagnosing the configuration procedure for a specific product from a complexity perspective in order to re-design the procedure. Once the administrator has completed the procedure, the procedure capture data are input to the Integrated Complexity Analyzer, which calculates the complexity scores from the input data and displays the complexity scores on a per-activity basis by means of the graphical analyzer. This data can then be viewed and interpreted by the consumability architect of the product/procedure so that he can identify complexity hotspots and re-design the procedure, which is then again carried out by an administrator. The differences in the complexity scores for subsequent runs of the procedure reflect the quantitative improvements in terms of complexity.

4.2 Tooling Components

In addition to addressing the requirements for two fairly different usage scenarios, the architecture of the Integrated Complexity Analyzer needs to be adaptable to changes in the model and the summary scores. This is needed because the model is continuously being refined, based on the results we obtain by running a variety of scenarios. Traditionally, this is a major challenge, because the model and its measures are at the heart of the overall system, and any change to the model ripples through the data structures that are evaluated by the Complexity Scorer to compute the complexity scores. In order to mitigate the impact of changes, we have decided to isolate the representation of the data (both input and output data) as much as possible from the complexity scorer logic. A thorough modularization of the architecture and a combination of several technologies turned out to be particularly useful in order to accomplish this. We will describe each of the components of the Integrated Complexity Analyzer along with the technologies that have been used in their implementation.

We use an 'XML-centric' approach (as changes to the complexity model are introduced into the overall system by a change of the XML schema) and rely on the Eclipse Modeling Framework (EMF) [3] to automatically generate the Java objects that correspond to the elements in both XML schemas that represent the incoming complexity data and the complexity scores, respectively (depicted in the center of Figure 3). By doing so, every time a new element or attribute is added to an XML schema, we simply re-generate the EMF objects for the XML schema in which the change occurred. While a corresponding EMF object

– a Java class with accessor methods – representing the new XML element is generated (for which new code needs to be written in the complexity scorer), already existing EMF objects remain unchanged. An additional advantage of EMF is that we obtain the XML parser 'for free', as appropriate Java classes to serialize/deserialize XML files into/from EMF objects are automatically generated by EMF. Seamless transformation between XML, UML and relational database schemas based on Java is the core purpose of EMF.

The *Complexity Scorer* – the core component that summarizes and scores the complexities of both processes and procedures according to the measures described in the previous section – is implemented in Java and inputs the complexity data that is represented as an EMF model. The complexity scorer needs to distinguish between the two scenarios described above as their complexity data is fairly different. The XML schema for input data specifies a flag indicating whether the data in an input file refers to either procedure capture data or to an exported process model. The calculated complexity scores are represented as EMF objects, too.

The remaining three components of the Integrated Complexity Analyzer leverage various EMF extensions. The *Plot Generator* inputs an EMF model containing the complexity scores and transforms them into a graphical representation (typically a bar chart whose various display options can be selected by the user). Flexible support for graphic widgets is provided by the eclipse plugin for Scalable Vector Graphics (SVG) [11], which can be directly displayed in the Mozilla Firefox v1.5 web browser. The *Process Tagging* component works off the EMF model containing the complexity scores and serializes them by means of EMF into the XMI format so that the classifier-tagged process model is output in a format that can be directly consumed by the WebSphere Business Modeler tool. Finally, for each analysis, both the raw complexity data that is input into our system as well as the complexity scores computed by the Integrated Complexity Analyzer are persistently stored in a hybrid relational/XML database. We use an early adopter version of the IBM DB2 Universal Database Enterprise Server Version 9.1. The *Data Access Objects* implement persistent storage for EMF models in a relational database based on Service Data Objects (SDO) [8]. The advantage of storing structured EMF objects over storing a set of text documents merely as character large objects (CLOBS) in an RDBMS improves their retrieval significantly: as the data remains structured, one can easily issue queries of the type 'retrieve all the actions in a procedure/process whose memory complexity is greater than X' against the database.

5 Evaluation

In this section we consider a sample subprocess of ITIL Change Management: the process accepts a change request and updates the corresponding configuration items (CIs). A change request results in the creation of new CIs or the modification of existing CIs. As part of the process, it is necessary to extract information referring to the CI from the change request, authorize and validate the changes

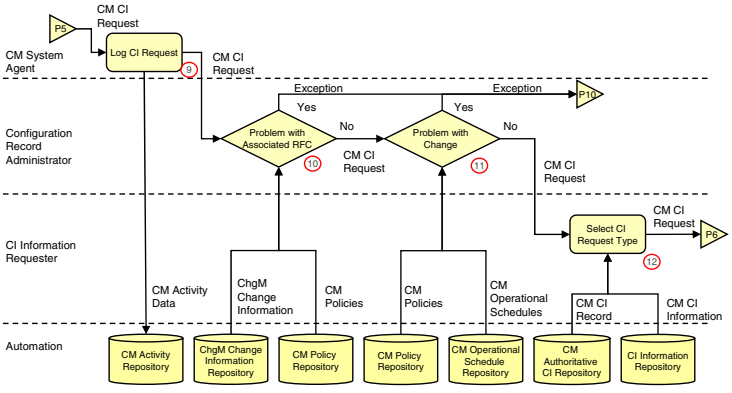

Fig. 4. Partial workflow for a change management subprocess

against policies, issue necessary queries to extract other needed CIs to accommodate the request, request modification of the CM Authoritative CI Repository design if needed (for new CI types), either create or update the appropriate CIs, and finally report the results of the CI changes in a history repository.

The sample subprocess consists of 27 tasks; a part of the flow is depicted in Figure 4, which includes the tasks numbered from 9 to 12 that are carried out by four different roles: Change Management (CM) System Agent, Configuration Record Administrator, Configuration Item (CI) Information Requester, and Automation (with a set of data repositories and services).

The per-task complexity is computed based on the complexity metrics introduced in Section 3. For example, task 11 is a decision point which checks if a requested change is allowed by the policies and execution schedule. It generates either an exception or proceeds to task 12 if the change can be carried out.

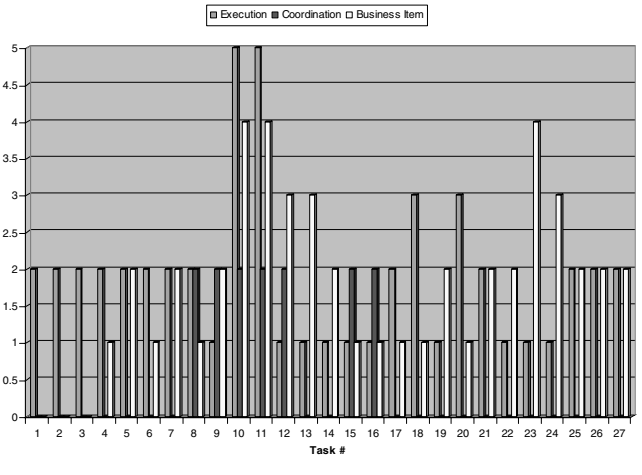

Fig. 5. Process complexity per-task view for a change management subprocess

Table 1. Process-wide complexity metrics

Complexity Measure	Metric	Value
Execution	Number of Tasks	27
	Number of Decision Points	11
Coordination	Number of Shared Tasks	0
	Number of Meetings	0
Business Item	Number of Business Items	10

Task 11 involves one role, four business items, two decision branches, and one coordination link with a different role (CI Information Requester).

Once the per-task metrics have been computed, they can be aggregated to produce process-wide views to identify the complexity bottlenecks within this process, or process-wide metrics to facilitate cross-process comparison. Per-task views are graphs showing all per-task metrics in bar charts. Figure 5 provides a per-task view for all 27 tasks. The x axis indicates the tasks, and the y axis indicates the metric values. All per-task metrics can be plotted separately, or aggregated for three high-level views of execution complexity, coordination complexity, and business item complexity. The per-task metrics can be analyzed to pinpoint complexity bottlenecks. For example, Figure 5 indicates that task 10 and 11 are relatively more complex in comparison to the other tasks in the process. On the other hand, this process does not have significant complexity spikes. The overall process complexity metrics are summarized in Table 1.

6 Conclusions and Outlook

In this paper we have proposed an approach to quantifying complexity of IT management processes. We model a process as a set of roles, each of which may participate in a set of tasks that either consume or produce business items. Moreover, process complexity is quantified from three dimensions: execution complexity with regard to the level of automation and decision making, coordination complexity with respect to the coordination links and the complexity of shared tasks, and business item (BI) complexity representing the source score for supplying values into the business item. Finally, we have described the design and implementation of the tooling we implemented, and have evaluated an IT management process to demonstrate the usage and applicability of the approach.

The benefits of our approach are as follows: First, the complexity analysis results guide the IT process reingeneering effort and identify activities that should be delegated to automation. Second, the IT process complexity model and its measures establish the basis for measuring improvements between two versions of an IT service management process. Third, the approach is applicable to different stages in the lifecycle of a process: while its core focus is on providing a quantitative framework for evaluating processes in the design stage, it can be applied to an already deployed process as well, in which case the measurements obtained during its execution apply to a specific flow through the process.

While these initial results are encouraging, there are several areas of further work: As an example, we are currently working on a mapping from the measures in this paper to higher-level measures such as success probability, configuration time, labor cost, and required skill level to complete configuration tasks. We are also exploring more service management process examples such as problem management and release management, and studying its applicability to business processes, e.g., Information and Communication Technology (ICT) business management processes for communication services, fulfillment, and billing.

Acknowledgments

The authors would like to express their gratitude to Aaron B. Brown for his work on an early version of the complexity model and to Christopher Ward for his assistance in obtaining a variety of IT Service Management Process Models. Both are employed by IBM. In addition, we are indebted to the reviewers for their helpful and constructive comments that helped us improve the quality of this paper.

References

1. IT Infrastructure Library. ITIL Service Support, version 2.3. Office of Government Commerce, June 2000.
2. A.B. Brown, A. Keller, and J.L. Hellerstein. A Model of Configuration Complexity and its Application to a Change Management System. In A. Clemm, O. Festor, and A. Pras, editors, *Proc. of the 9th IFIP/IEEE International Symposium on Integrated Management (IM 2005)*, pages 631–644, Nice, France, May 2005. IEEE.
3. F. Budinsky, E. Merck, and D. Steinberg. *Eclipse Modeling Framework*. Addison-Wesley, 2nd edition, 2006.
4. A.L. Couch, N. Wu, and H. Susanto. Toward a Cost Model for System Administration. In D.N. Blank-Edelman, editor, *Proc. 19th Large Installation System Administration Conference (LISA '05)*, pages 125–141, San Diego, CA, USA, December 2005. USENIX.
5. C. Kubicki. The System Administration Maturity Model – SAMM. In *Proc. 7th Large Installation System Administration Conference (LISA '93)*, pages 213–225, Monterey, CA, USA, November 1993. USENIX.
6. Ming Li. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.
7. D. Patterson. A Simple Way to Estimate the Cost of Downtime. In A.L. Couch, editor, *Proc. 16th Large Installation System Administration Conference (LISA '05)*, pages 185–188, Philadelphia, PA, USA, November 2002. USENIX.
8. Service Data Objects. `http://www.eclipse.org/emf/sdo.php`.
9. Geoff Tennant. *Six Sigma: SPC and TQM in Manufacturing and Services*. Gower Publishing, Ltd., 2001.
10. IBM WebSphere Business Modeler. `http://www-306.ibm.com/software/integration/wbimodeler`.
11. World Wide Web Consortium, W3C Recommendation. *Scalable Vector Graphics (SVG) 1.1 Specification*, January 2003. `http://www.w3.org/TR/SVG/`.