7.0 Introduction
The engineering in Document Engineering implies a systematic application of intellectual and technical knowledge to create tangible end products with economic or social value.
The essence of Document Engineering is a set of analysis and design techniques that yield robust and reusable models of documents and their roles in business processes. The discipline of an engineering approach ensures that these models are complete, useful, and reliable when realized as the data format for integration or as the interface and process specification in document-centric or service-oriented applications. The practicality of an engineering approach and a heavy reliance on design and implementation patterns ensures that the models can be developed and deployed at acceptable cost in a reasonable time.

In this chapter we’ll introduce a complete view of the phases involved in Document Engineering. In Chapters 8-15 we’ll discuss each phase and its constituent activities in detail, using experiences from a real-world case study that typifies the struggles of enterprises with incompatible models and applications.

7.1 An Approach, Not a Methodology
Our goal here is not to define a formal methodology. We aim more modestly to present a coherent and pragmatic approach for modeling documents and services that provide solutions that are practical and effective.

Three key factors shape the concepts and methods we present as the Document Engineering Approach:

1. **End-to-end scope.** We must be able to describe the information content and processes in a document exchange, identify the context of use and its relevant requirements and constraints, analyze and design a solution, and then implement and deploy that solution. Furthermore, we must expect that the requirements and constraints will change, so our solution must be evolvable. It would be pointless to develop a solution that can’t be adapted to changing environments, no matter how theoretically elegant or powerful it might be.
2. **The breadth of documents that we must be able to analyze, design, and implement.** In Chapter 1 we introduced the Document Type Spectrum that spans from narrative, publication-style documents to transactional, data-intensive ones. These contrasting types of documents have traditionally been analyzed and designed using substantially different approaches, which we unify by emphasizing what they have in common.

3. **The requirement that document exchanges must be implementable in a loosely coupled, technology-independent manner.** It is a fundamental principle of distributed and service-oriented architectures that the relationships between organizations or service providers must be adaptable and flexible because only the document interfaces are visible. It is neither necessary nor desirable for each party to know anything about the implementation on the other side of the exchange.

What each side needs to know can be completely captured using two types of modeling artifacts, models of the documents exchanged and models of the business processes, collaborations, and transactions. The latter form the context and specify the patterns and sequencing for the exchanges.

These three factors of scope, breadth of document types, and loose coupling, provide an approach that is flexible and heuristic. We present the tasks of Document Engineering in sequential phases, but within and between each phase, activities often overlap or repeat, and not every activity is required in every effort. However, presenting them in a typical progression makes them easier to introduce and better motivates the modeling artifacts that organize and visualize the most important results of each activity.

### 7.1.1 Modeling Methodologies

Modeling is inherently difficult. Modeling documents and the processes that use them is harder still because they can be both informal and abstract, giving us little to grab onto when we start. There is no single correct way to create these models, and many different methodologies have been proposed. They share the common goals of defining the
requirements of the context of use and communicating this understanding in one or more modeling artifacts.

<table>
<thead>
<tr>
<th>There is no single correct way to create document and process models.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every modeling methodology proposes a set of modeling activities. They may differ in the order in which the activities are carried out or how prescriptive they are about the activities and descriptions of their results.</td>
</tr>
<tr>
<td>How the models are described reflects the metamodel adopted by the methodology (see Section 3.3.1). Metamodels define the kinds of information that models contain, so more prescriptive metamodels increase the consistency among models, which in turn more easily exposes patterns within collections of models. Common metamodels also provide a useful basis for libraries of reusable patterns because the models they contain can be interpreted by anyone or any application that understands the metamodel.</td>
</tr>
<tr>
<td>For example, the UN/CEFACT Modeling Methodology (UMM)(^1) proposed a metamodel that specifies a set of progressively more detailed views of business processes, refining a high-level, goal-oriented view of the specific transactions in which documents are exchanged. Similarly, the RosettaNet initiative has defined a metamodel for describing supply chain processes called the Partner Interface Process (PIP).(^2) The UMM and RosettaNet metamodels were used as the foundation for the ebXML Business Process Specification Schema (BPSS), which has strongly influenced how we think about business collaborations that use web services.(^3)</td>
</tr>
<tr>
<td>We use aspects of these metamodels in Document Engineering, but we aren’t prescriptive about it. We advocate a less prescriptive and more pragmatic modeling approach to be consistent with the loosely coupled architecture of the Internet on which we expect that most of the models developed using Document Engineering will be deployed. Indeed, we are somewhat reluctant to use the term methodology when describing Document Engineering. We prefer to call Document Engineering an approach that can be followed to exploit the potential of extended enterprises, service-oriented architectures, and web services that embody the principle of loose coupling through document exchanges.</td>
</tr>
</tbody>
</table>
This approach embodies the principle of loose coupling through document exchanges.

When the participants in a business relationship can be on opposite sites of the globe, it is impossible for one side to impose a modeling methodology on another or audit the techniques the other uses to build a model. Once the model is defined and communicated, the process taken to develop it is invisible. So the methodologies each enterprise or organization uses to design its business processes and their associated documents should be as loosely coupled as the document exchanges. If only the document interfaces are visible, there is no justification for imposing a prescriptive modeling and design methodology. All that matters is whether the parties can produce and consume the expected documents or models appropriately.

Once the model is defined and communicated, the process taken to develop it is invisible.

As web services and service oriented architectures become more ubiquitous, businesses will expect reductions in the costs to evolve and operate new business models and the implementation technologies. This implies a pragmatic approach to modeling: one that is efficient, does not require nonessential activities to achieve the desired result, and, most importantly, actively strives to build on previous efforts.

7.2 The Document Engineering Approach

In Chapter 3 we introduced the model matrix (Figure 3-7) to depict the relationship between the different kinds of models enterprises use to arrange their activities. The models we organize in the upper left corner of the matrix are broad in scope and abstract in perspective. As we move to the right and down in the matrix, models become narrower in scope and more concretely tied to technology and implementation. We are now ready to introduce the Document Engineering approach as a set of activities to create and reuse these models.

7.2.1 Unified Approach to Modeling

Document Engineering relies on the skills and tools of business process, document, data, and task analysts. One of the innovations of Document Engineering is to exploit these
different techniques for reaching the same goal. That goal is developing models that are abstract enough to be reused as patterns but concrete enough to be implemented. Figure 7-1 graphically depicts this common goal as reaching the middle of the model matrix from different starting points.

**Document Engineering exploits different analysis techniques for reaching the same goal.**

![Figure 7-1. Analysis Perspectives Used in Document Engineering](image)

Business analysis typically starts with abstract views of business models and processes, which are organized in the upper left corner. This high-level analysis establishes the context for understanding the semantics of the information in the other sections of the matrix.

Task analysis (or user analysis) is the observation of people performing the tasks or *use cases* when the application or system must support human interfaces and not just
other applications. Task analysis identifies the specific steps and information that people need to carry out a task, so it is based on actual artifacts and activities, which are represented on the right side of the matrix. Task analysis and document analysis are closely related; document analysis reveals candidate information components and task analysis reveals rules about their intent and usage. Task analysis is especially important when few documents or information sources exist, because human problems or errors can suggest that important information is missing.

Document analysis tends to start from analysis of document instances. We show this on the lower right side. These techniques extract or disentangle the presentational, structural, and content components of documents or other information sources.

Data analysis (or object analysis⁴) techniques often start from a conceptual perspective about a domain and yield an abstract view of the information components revealed by document analysis. So this approach is represented as starting from the lower left corner of the model matrix.

Modeling methodologies are a means to a desired end—the set of modeling artifacts produced by the analysis and modeling effort. These artifacts include documents, their models, and libraries of reusable patterns used in models. We advocate a stronger focus on these artifacts than on the means for creating them. We call this an artifact focused view of modeling.

### 7.2.2 Artifact-Focused View of Modeling

The artifact-focused view of modeling concentrates on producing modeling artifacts and on reusing existing modeling artifacts when they fit. We’ll explain this approach by using an analogy with the income tax systems that operate in most countries.

A formalized methodology is equivalent to the entire tax code. Because the tax code has to be comprehensive, it is generally too complex for people other than tax attorneys and accountants to use. So the taxation authority provides forms that assist taxpayers by organizing the information they need to provide to comply with the tax code rules. These forms are themselves supported by booklets (or guidelines) that provide instructions for
filling them out. Finally, software vendors have developed applications (such as TurboTax) that guide taxpayers through the process of filling out electronic forms and perhaps even submitting them over the Internet.

When most people pay their taxes, they focus on the tax forms because these are the artifacts required by the taxation authority. They may consult the guidelines or even the tax code, but only on an ad hoc basis. They certainly don’t start by reading the tax code and all the guidelines from beginning to end. Likewise, the taxation authority doesn’t care about the process the taxpayer or accountant followed to fill out the forms; only the numbers matter.

We are proposing the same approach in Document Engineering. We provide lots of guidance and explanation to help you create models of documents and processes, but we emphasize the models. By focusing on the artifacts rather than the methodology by which they are created, we give the document engineer more flexibility in capturing relevant data or metadata about documents and processes whenever they arise. This is also a key benefit of most tax software applications, which allow the various subparts and forms to be filled out in almost any order, pulling the information together at the end to calculate any required payments or refunds.

### 7.2.3 The Modeling Phases, Tasks, and Artifacts

Figure 7-2 depicts the Document Engineering approach as a path through the model matrix to carry out a set of analysis, assembly, and implementation tasks. We show this path as being equally wide as it winds its way through the phases of Document Engineering, but in practice different phases may get more or less emphasis, depending on the management and strategy decisions that shape the project. Top-down or strategic efforts to align business organization and technology cut a broad swath through the top of the model matrix. These efforts create models that are very abstract or very generic, partitioning activity into large, goal-oriented chunks to provide a big picture view of the context of use.

In contrast, bottom-up and more document-driven projects emphasize the path through the lower half of the model matrix. These efforts may yield a large number of models for
transactional processes, often refined by the specific types of document they produce or consume.

But high-level goal-oriented models lack the detail needed for implementing and integrating the applications built to achieve them, and low-level models of documents and information components by themselves don’t provide much help in aligning high-level business goals with technology choices and implementation decisions. That’s why it is worthwhile to follow the entire path through the matrix. Developing a variety of models of varying emphasis and granularity ensures that any new models we create for business processes and documents are complete, consistent, robust, and deployable in applications that meet actual business requirements.

Following the complete path also helps to overcome the fundamental modeling challenge of achieving a consistent level of abstraction so that patterns and models from different perspectives can fit together. There is a large granularity gap between business models and document models. Our path through the model matrix yields successively more granular models that bridge the gap.
Figure 7-2. Phases of the Document Engineering Approach

On the top row we begin with high-level, organizational analysis to understand the main business activities and the people and organizations that participate in them. This strategic perspective is an essential foundation for developing a service-oriented architecture or carrying out mergers or acquisitions. For example, when HP and Compaq merged, teams of executives and business analysts spent months analyzing how each enterprise did business to decide what practices, divisions, products, and people should be retained. Models at this level, called the business domain view in ebXML, describe the broad context of use for the documents and processes we will define at more granular levels.

What we call context is the collective sum of the requirements for our project. So the first phase of Document Engineering, “Analyzing the Context of Use,” involves identifying strategic business objectives in terms of business model requirements and the rules they must satisfy.
Understanding the business process is important for expressing the context of use. In business process analysis we often contrast the way things are with how we would like them to be. After we describe an As-Is model, we can improve its processes by applying existing patterns or we can invent completely new ones.

We create these process As-Is models in the “Analyze the Business Process” phase.

As we saw in Chapter 4, much of what businesses do can be described using a small repertoire of business patterns. For example, we might decide that our context of use fits the Procurement pattern, and that its processes and information exchanges describe the transactions and documents required. Or we may best describe our model as an incremental information trail in which documents are created, consumed, added to, and subtracted from as they are passed along from one process to the next.

Choosing and instantiating appropriate patterns for business processes entails adopting a predefined context of use. Using a business process pattern also suggests the relevant users and other stakeholders from whom we can obtain or confirm requirements.

Choosing a pattern for a business process entails adopting a predefined context of use.

So designing business processes can be best described as “Applying Patterns to Process Models.”

Describing the actual documents needed by a business model starts to take place during the “Analyze Documents” phase. 5

The To-Be process model identifies the roles that documents will play; document analysis exposes the specific business rules that govern the content, structure, presentation, syntax, and semantics of the information contained in the documents.

We analyze existing document models (such as XML schemas) as well as any implementation guidelines and standards, sample document instances, web pages, and other information sources to harvest all potentially meaningful information components and the constraints that govern their values, arrangement, and use. Of course we can’t ignore the people who create, review, approve, query, or do other things with these documents. In particular, in domains or new business models where few documents exist,
what we can learn from people is critical because we can derive information and
document requirements from their goals and tasks. In many situations existing documents
are extremely valuable proxies for, or confirmations of, what people tell us.

The “Analyze Document Components” phase starts with the harvesting task. This
identifies the individual semantic components contained in each of the selected
documents or information sources. Part of this rationalization involves choosing
meaningful names for the components. But naming is inherently a contentious and
iterative task, so the names are tentative at this point.

In “Assemble Document Components” we assemble sets of these information
components into meaningful structures to create a coherent conceptual view we call the
document component model. We advocate doing this by using data analysis techniques
that normalize the components into structures based on their functional dependency.

We then turn from analysis to design as we start to create models for new types of
documents based on the components, structures, and associations in our document
component model. We call these new models document assembly models. In the
“Assemble Document Models” task we apply the rules for assembling the information
components necessary for each different type of document required for the given context
of use.

Effective design involves the analysis, reuse, and creation of patterns.

Effective design of processes and documents requires us to recognize when a pattern
can be reused, when a new pattern should be created, and what distinguishes one pattern
from another. So the conceptual document assembly models and process models we
develop can reuse the patterns from a variety of internal and external sources.

7.2.4 Implementing Models in Applications
These conceptual models represent substantial investments in understanding sets of
business rules and capturing contextual requirements. In the implementation tasks, we
create modeling artifacts to define and drive applications. We use these in an explicit way
to implement a solution in an automated or semiautomated manner, what we call a model
based application. In other words, we exploit the conceptual models to bridge the gap between knowing what to do and actually doing it.

**We exploit conceptual models to bridge the gap between knowing what to do and actually doing it.**

Model based applications are often implemented using software whose generic functionality is made context-specific by configuring or extending it to use the context-dependent information and behavior specified in the model. The first step in achieving this is to realize the conceptual models in a suitable language.

For documents we call the realized artifact, the document implementation model. Document implementation models realized in markup languages are more commonly known as schemas, as we saw with XML in Chapter 2. For example, when XML is used to encode document implementation models, many aspects of the integrity of a document’s information components, as well as the business rules applied to the data, can be derived from the XML schemas. (See the Sidebar).

**SIDEBAR: Is Document Engineering Compatible with User-Centered Interface Design?**

The Document Engineering approach for design and implementation of user interfaces might at first seem incompatible with the conventional user-centered or usability engineering approach to interface design. The latter approaches rely heavily on iterative prototyping and evaluation with an outside-in perspective in which document and process models are less explicitly considered. In contrast, the model-based approach of Document Engineering is more inside out, is especially appropriate for designing systems rather than user interfaces, and defers user interface design until the models are mostly complete.

Nevertheless, we think that Document Engineering is both compatible with and complementary to usability engineering, especially for applications enabled by web services and document exchanges. We certainly aren’t claiming that document implementation models can directly determine every aspect of user interface design. But models can:
• Define the information that needs to be presented in user interfaces.
• Serve as hypotheses or checklists to help user interface designers determine the optimal presentation and interaction structure for applications used by people.
• Generate prototype interfaces.
• Enforce rules or best practices about user interface design along the way.

Likewise, functions that control the transaction workflow or application logic can often be derived by analogy from models of business process. What is important is that these models remain loosely coupled with the user interface design. Further transformations that apply presentation rules to document instances can also be implemented to meet formatting or other rendering requirements for different classes of uses or devices.

For models of business processes, realization means adopting a suitable metamodel (such as the ebXML BPSS) to encode the specific rules and the requirements for our given context of use. This means that the modeling artifact itself is encoded as a document. We call this realized artifact, the business process implementation model. For example, RosettaNet PIPs are examples of business process implementation models encoded in XML. Web services and service oriented architectures can be implemented in this model-based way when the document and process models they use are designed to separate generic and context-dependent functionality.

7.3 Analyzing the Context of Use
The universe of discourse for any business or Document Engineering project is potentially a vastly complex set of processes and information components. We cannot hope to analyze its entirety. By necessity we must define a subset of the most important components and processes. This isn’t always easy because, to misquote John Donne, “No document is an Island, entire of itself.”

A typical large organization or firm might use dozens of different types of documents with complex organizational and process interdependencies. For example, information on
an Order document can be connected in some way to account ledgers, sales reports, catalogs, production schedules, advertisements, user manuals, and numerous other documents. Furthermore, the relationships among these different types of documents vary for different information components. A Product Description may relate to catalogs and advertisements, but Buyer Party would relate to sales reports and accounts ledgers. It is not hard to see how the scope expands very quickly, and we must explicitly identify what is in and out of scope to make sense of any of it.

### 7.3.1 Requirements

Identifying requirements in the initial stages of a project reduces the likelihood that you’ll have to throw away or rework the analysis, design, and implementation phases that follow. But too often people view technology adoption as a requirement rather than requiring a business need to justify the adoption of technology. A focus on new technology and new ways of doing things often suppresses consideration of important legacy technology or existing business processes. The collapse of the dot-com bubble should have taught us lessons about putting technology before business considerations and ignoring the core requirement of making enough money, but if we’re not careful we’ll forget it all in the next waves hype about web services, RFID, ubiquitous computing, and whatever comes next.

Requirements are constraints on possible solutions that must be satisfied for the solution to be acceptable. They are most often expressed as functional descriptions (or rules) of what the solution must do. But they can also include performance characteristics, quality attributes, or conformance to regulations or standards. In other words, requirements are the way we express the context of our document exchanges. Of course, rules and requirements will continue to emerge throughout the project, but identifying as many requirements as we can is a pretty good starting point. Identifying can categorizing requirements are the main topics in Chapter 8, “Analyzing the Context of Use.”
Many of these requirements will be expressed as rules about the content, structure, and presentation of documents and their components. We use these to identify and design new types of documents. Other requirements will be expressed as usage rules or policies about access to information or control of its processing. We use these to formalize the definitions of the context in which the documents are used.

**Collecting requirements and rules is a heuristic and iterative exercise**

Collecting these requirements and rules is a heuristic and iterative exercise that requires us to take the complementary perspectives of the archaeologist and the anthropologist. Like the archaeologist, we search for artifacts and try to interpret them even though the organizations or people who created them might be extinct and no longer available to help. We might discover legacy formats and paper documents whose processes have been frozen in time. But these artifacts might refer or link to other artifacts, and slowly we begin to understand them. And like the anthropologist, we locate people who work with the artifacts, and they may refer or link us to other people, who help us find more artifacts and people.

Sometimes requirements don’t emerge until after the first version of a solution is implemented. If we anticipate this and implement the application in a loosely coupled and model-based manner, we can view new requirements as good news rather than bad news.

Business analysis and task analysis techniques such as user interviews and questionnaires can tell us how people think they use information and documents. But because people sometimes tell us what they think we want them to tell us, we can’t always take what they say at face value. That’s why the best way to gather requirements is to balance the artifact-driven work of the archaeologist with the perspective of an anthropologist studying people and phenomena in their natural surroundings. We must observe and listen carefully to learn about what processes and documents exist, where they can be found, how they are used, how useful they are, and the rules that govern their use. But we must be on the lookout for errors and inconsistencies in what we hear and discover and sometimes let the documents speak for themselves.
In this book we represent the models we develop during these activities in artifacts such as UML use case diagrams and descriptive worksheets.

**7.3.2 Patterns of Context**
In Chapter 3 we defined patterns as models that are sufficiently general, adaptable, and worthy of imitation that we can use them over and over. We also described how businesses follow patterns because of common requirements for their specific processes, geography, products, legal environment, business models, and so on. So we can see each combination of these factors as a common pattern of requirements.

We use the context of use to organize and analyze requirements and rules.

Consideration of the generic procurement pattern of one buyer and one seller will ultimately be situated in a richer context that specifies the industries in which the procurement takes place, goods being procured, locations of the buyer and seller, regulations or laws governing the activity, and so on. These dimensions of context are a common way of classifying patterns of business rules and requirements. In other words, we use our appreciation of the overall context of use to organize and analyze the requirements and rules that will drive our design of processes and their documents.

**7.3.3 Scope of Context**
To understand the scope of the processes in our business model, we need to recognize dependency relationships among business processes. Dependency describes the impact that a change to one object has on another object. We say that if a change to A inherently changes B, then B is dependent on A.

Many businesses processes are completely independent. For example, hiring a new employee doesn’t affect the product catalog, but adding a new product to the manufacturing schedule may change the catalog, and changes to the catalog may change the structure of sales orders. So we could say that while hiring is independent of marketing and sales, marketing and sales are dependent on manufacturing.

The organizational charts and policy or procedure manuals we discussed in Chapter 4 are examples of documents that can suggest dependencies within an enterprise. But often
these are descriptions of what the dependencies are supposed to be rather than what the people or systems actually do.

The more legitimate dependencies we encounter in our project, the more patterns for reuse we will find, but at the cost of greater complexity. So we should aim for a scope that includes enough dependencies to ensure that the context is realistic, but not so many we cannot comprehend the interactions among them. Understanding the patterns involved can help us with this. For example, suppose we identify a pattern that describes the business processes that interests us. When applied in a specific industry, with specific firms, in a geopolitical environment, and so on, the pattern is inevitably customized to meet the requirements of a more concrete situation. Seeing how a pattern has been adapted for different contexts of use can help us understand potential dependencies.

7.4 Analyzing Business Processes
We define a business process as a chain of related activities or events that take specified inputs, add value to them, and yield a specific service or product that can be the input to another business process.

Some business processes are conducted entirely within a firm and are called internal, enterprise, or private business processes. In contrast, external, collaborative, or public business processes are carried out between two or more business parties. The Document Engineering approach applies to both kinds of processes but is especially useful for the latter, where different implementation technologies mandate the loosely coupled architecture of document exchange.

Of course, private business processes and public ones overlap because they must connect if a firm is to do business with any other party; the business exists to exploit the results of its private business processes in its public ones. This intrinsic connection between the private and public processes establishes a dependency between them.

**Businesses exist to exploit the results of their private processes in their public ones.**

But it can be difficult for processes to span the boundary between two firms, because of the inherent flexibility in how abstractly processes are described. Two businesses
might use different levels of abstraction or granularity to describe the processes they need to connect, making their process descriptions incompatible. Chapter 9 explains two ways in which we can avoid this problem:

- Use the concepts and components provided by a business reference model, whose hierarchical organization of processes has been rigorously designed to reinforce granularity.
- Express all process models at the granularity where we can identify the documents that they produce and consume.

Chapter 9 also introduces the modeling artifacts we typically produce during the Business Process Analysis phase. These include worksheets that help us organize the information we learn about processes and UML activity diagrams, UML sequence diagrams, and other forms of flowcharts that record the processes once we understand them. These diagrams depict the structure of collaboration and interaction between the people and services implied by the pattern. These models also begin to express our understanding of both the transaction semantics and the required contingencies between each document exchange.

### 7.5 Designing Business Processes

The relationship between public and private processes is complicated by the tension every business faces in balancing the benefits of designing documents to accomplish its private processes in an optimal way with the need for its processes and documents to be understood by other businesses. This tension induces businesses to reuse existing models for processes and documents whenever possible and encourages them to design new models that encourage reuse by others, even if doing so results in a less-than-perfect solution for its internal needs. This is why the task of designing new business processes often involves more adopting and adapting of patterns than invention of something new. Much of business process design is actually more like reusing existing business patterns. This is the perspective we take in Chapter 10, “Designing Business Processes with Patterns.”
This tension between public and private processes induces businesses to reuse existing models.

As we saw in Chapter 3, when identifying reusable patterns it is desirable to describe the models in conceptual rather than physical terms. Business process libraries are a useful repertoire of business process patterns precisely because they do not dictate specific technologies for implementation.

Even if a business process pattern is only a partial fit to a particular context, it can still provide useful insights for identifying further requirements. For example, some aspects of the procurement pattern are the same whether one is buying steel, paper clips, or university courses.

7.6 Analyzing Documents

The objective of document analysis is to create a conceptual model that encompasses all the information requirements within the required context of use. We start this phase by determining what documents and information sources we need to analyze. This is the primary topic of Chapter 11, “Analyzing Documents.”

7.6.1 Create the Document Inventory

A document inventory is the collection of documents and related artifacts we analyze. The inventory lists the sources we identify along with metadata about their purposes, origins, and other attributes that will help us select a subset to analyze in detail.

The richer the document inventory, the more effective any analysis will be.

Because the inventory exposes both information components and related business rules, the richer this inventory, the more effective any analysis will be. So we need to take a broader view that goes beyond traditional printed-paper documents and their electronic analogs.

In fact, much of what we need to analyze may not be in a traditional document form; much of it may look more like sets of data. So we stretch the meaning of document to
include information in databases, spreadsheets, and accounting systems, as well as catalogs, brochures, schedules and calendars, word processing files, and web pages.

And not all information requirements are necessarily recorded in documents themselves. There may be useful metadata about documents and their components in the form of document definitions, data models and schemas. Additional metadata can be found in style guides, industry standards for the domain, application interfaces, and artifacts from previous studies and analyses.

Last but not least, the inventory should include any undocumented information from the people involved in the exchange of documents. The people who create and use the tangible parts of the document inventory can tell us what to look for and where to find it. They also have much tacit knowledge that they subconsciously understand and apply, and we can encourage them to make it explicit by engaging them in conversation, or we can learn it implicitly by observing them at work. We’ll need their help to understand the documents and information sources we find. And before we’re through, we’ll want them to review the proposed models and their embodiment in applications.

7.6.2 Sample the Document Inventory
It is unlikely that we’ll be able to analyze everything in the inventory in detail, so we need to take a representative sample. Determining what sources are representative is another iterative task, because we won’t know the size and variability of the inventory until we’ve collected it. The people who provide the documents might be able to help, but they often have a limited or inaccurate understanding of business processes and documents other than their own.

Not everything in the inventory is equally valuable. We may also want to emphasize or give more weight to documents that are especially important or authoritative, but we won’t always know this at first either. For example, a document model in the form of a schema or its proxy (such as a data entry form) may give more and better information about requirements and constraints on components than an individual document instance can. However, a set of several sample document instances may prove more suitable for identifying additional business rules.
In general, the more document instances we consider when we do our analysis, the more precisely we can recognize the business rules that express requirements. Obviously, if the instances are homogeneous we need fewer of them than if they are not. This means there is a law of diminishing returns that decides when we have seen enough instances; as long as new instances expose new rules, we’re probably not done.

7.7 Analyzing Document Components

After we select a representative sample from our document inventory, the next activity is to isolate any semantic components they contain. We call this harvesting the components. This phase is discussed in detail in Chapter 12, “Analyzing Document Components.”

7.7.1 Harvest the Components

There are two distinct tasks involved in harvesting: separating the underlying meaning from presentational components and disaggregating existing structures. We can illustrate both tasks with some examples.

In Chapter 6 we described some of the interoperability challenges that GMBooks.com faces if organizations can order books either by filling out a form or by having an application order them using electronic documents (see Figure 6-2). These two different physical representations of Orders must both contain the same content components or they won’t be acceptable to GMBooks.com. They both have content components such as the publisher supplying the books, the book titles and authors, the quantity ordered, details of the party making the order, the address to which the books are to be delivered, the instrument or mechanism by which the affiliate proposes to pay GMBooks.com, and so on. However, how these components appear can be radically different.

Separating meaning from presentation involves recognizing the stylistic conventions or presentational components being applied to information in its various formats. For example, the affiliate ordering books by fax might create the order using a word processor, sort the books by publisher, and center and underline the publisher’s name before each set of books. Another affiliate might send an electronic order for the same books using a procurement application that intermixes books from different publishers.
with no additional formatting. The books being ordered are identical, but the documents differ in the presentations assigned to their components.

In addition, the two orders may be arranged in different structural components. Some of these may be groups or composites of components to facilitate tasks like data entry or display. These presentational structures are usually required by people because business applications don’t care. For example, the components in an order may be organized into separate structures such as the order header, details, and summary, or in sections relevant to different parties or types of information. One set of components may be arranged in a structure labeled Delivery Address, another labeled Billing Address, and a third labeled For Office Use Only.

Presentational components like boxes, rules, shading, indentation, and other formatting devices can be used to reinforce these presentational structures.

**Presentational structures are often the most salient patterns in narrative documents.**

Presentational structures are often the most salient patterns in narrative documents because their individual content components are not explicitly distinguished; occasional exceptions when content components like Note, Warning, or Code Sample label parts of narrative documents prove the rule. It is more likely that narrative documents use structural terms like Chapter or Section as the primary organizational mechanism. Others use hierarchical numbering schemes that are correlated with presentational components like type face or size to reinforce the structural distinctions, as we do in this book.⁹

Identifying presentation components and presentational structures enables us to determine whether these stylistic characteristics are necessary to understand the information contained in the document. Differences in appearance of the two physical implementations may or may not affect the semantic equivalence of the information they convey.

As we analyze these components, we try to replace any required presentational components with content components. We also try to disaggregate any presentational
structures into their atomic information components, because that is the granularity necessary to identify their meaning and also to find reusable components.

While we treat presentational components and presentational structures as cues or clues for locating content components, that doesn’t mean we can discard them once we’ve located the content. There are often fidelity and integrity requirements for information components that we need to record as rules in our models. For example, we might have a presentational integrity requirement for preserving the original appearance of a document when it is reimplemented using different technology, such as when publishing printed articles on the Web. Similarly, we can use the presentational structures in which we find semantic components as design hypotheses about how to organize information into the most effective user interfaces for applications that display or collect information.

### 7.7.2 Name the Content Components

As we harvest candidate content components we give them names to distinguish them and suggest their meaning. These component names need to be unambiguous within their context of use.

Sometimes unambiguous names aren’t difficult to create, especially for transactional documents where each field on a form may already have a label. But in more narrative types of documents, where fewer components are explicitly distinguished, much of the time we have to use less obvious starting points. In addition, existing names probably won’t be consistent, so we should develop or adopt rules to make names precise and unique.

How rigorous we need to be with naming components depends on the size of the document inventory and the complexity of the project. At a minimum, we should maintain a component name dictionary, a list of the terms used in the names of components along with their definitions. In a more complex environment, we might find it necessary to use a controlled vocabulary or a formal ontology to improve the quality and consistency of component names.
Naming components is a contentious, iterative and ongoing activity. Naming is a contentious, iterative and ongoing activity throughout our analysis. As we identify candidate components, we should not be surprised if new components suggest changes to names already assigned. We should also not be surprised when user testing reveals that names that seemed perfectly sensible to us don’t make sense to the people who have to use them.

The primary modeling artifact from the analysis of information components is a Table of Candidate Content Components. This aligns the components harvested from all the document sources so that we can identify synonyms, homonyms, and semantic overlaps.

7.7.3 Consolidate the Candidate Components
At this stage in our analysis we consider all the components as candidates, because we don’t know if they are to be part of the final model or not. The way we confirm this is to establish their individuality by identifying what makes them separate components.

In other words, we need to merge any synonyms (components with different names and the same meaning) by selecting a single term to replace the different ones. And we need to split the different senses of homonyms (components with the same name but different meanings) by assigning more distinctive names to each one.

This consolidation activity merges the separate sets of candidate components we created from each source during the harvesting activity into a master or combined set. The modeling artifact we produce is called a Consolidated Table of Content Components.

7.8 Assembling Document Components
The result of the consolidation activity is a set of semantically unique components. The first step in creating models of documents from this set is to establish the required structures and identify any associations between them. This is the subject of Chapter 13, “Assembling Document Components.”
Put simply, we want to organize the set of content components we’ve created into structures like Address or Item that can be reused as building blocks in the more complex structures we know as documents.

But how many structures should we create for our content components? The optimal structures may not be obvious. If we don’t create any, we have unlimited flexibility in how we can assemble the individual components into documents. However, this flexibility would be inefficient and prevent us from recognizing patterns suitable for reuse because we would not be building in any of the lower-level dependency rules like those between the components aggregated into structures like Address or Item.

At the other extreme, we could create a few rather large or coarse aggregates like a Catalog Entry or an Order Details structure. It would be straightforward to reuse components of this size because most documents wouldn’t need many of them. But there would likely be substantial redundancy between each different implementation, leading to possible semantic confusion.

### 7.8.1 Formalize the Component Model

If we treat this problem of assembling components as an informal one and apply intuitive and heuristic techniques, it may be possible to come up with a set of structural components that let us build models of the document’s we need. In fact, we’ve used the example of Address and Item precisely because it is the kind of obvious and intuitive composite that would emerge from even an informal design approach.

When we have a very small set of candidate components in a controlled environment of limited scope, an informal assembly approach may be sufficient. But in most contexts an informal approach that creates structures because they seem reasonable isn’t likely to yield an optimal solution. Nor is it likely that any related projects would get the same or even compatible results.

More predictable results come from following more rigorous techniques for refining content components into aggregate structures.
More rigorous techniques for assembling structural components produce more predictable results.

We advocate an approach based on the concept of functional dependency. We first introduced the concept of dependency when we discussed analyzing context earlier in this chapter. Functional dependency is the principle behind a set of data analysis techniques collectively called normalization. These techniques are widely used by database designers to yield relational models that minimize redundancy and maintain information integrity. We have adapted these techniques to produce models of document components.

We call this modeling artifact a document component model but it may be more familiar to data analysts as a domain model.

This model presents an overall conceptual view of the all the information components required for a given context of use. It is convenient to represent this model as a UML class diagram. From this set of associated semantic structures we can assemble all our new document models that may span the transactional and narrative ends of the document type spectrum.

### 7.8.2 Associations between Structures

The document component model that emerges from our analysis does not describe a single document structure. Rather it defines a network of all potential document structures that might be required within our context of use.

For example, if our context of use involved procurement, we might identify structural components such as Order, Buyer Party, and Invoice. Each of these may have associations with the others. An Order may be placed by a Buyer Party, a Buyer Party may receive an Invoice, and an Invoice may reference an Order. These rules describe a networked set of associations with no defined start and end points. Their document component model would describe all these possible associations.

We design specific types of documents by organizing their structural components into what we call document assembly models. In effect, when we create a document assembly
model we are defining a specific path through this network of associations. This creates what data analysts may call a view. But which paths (or views) we choose for a specific type of document is an issue of design rather than analysis—and we’re not quite there yet.

### 7.8.3 Refine Component Names

During the consolidation phase we gave tentative names to candidate components. An important checkpoint at the end of the analysis tasks is to refine these names.

For example, now that we have organized content components into structures, the qualified names we may have assigned to eliminate ambiguity might now be redundant. Do we need to call the component Order Reference if it is part of the Order structure? Will Reference be an adequate name?

### 7.9 Assembling Document Models

A document component model is the capstone of the analysis work we carried out. It represents the As-Is model of things as they are. But our true goal is to find a way to make things better, and to do that we need to design new, To-Be documents. This is the subject of Chapter 14, “Assembling Document Models.”

In fact, it is often when we create a document component model that we start to formulate what that better way may be. Experience tells us that the analysis involved in creating a document component model gives us a deeper appreciation of the rules and requirements of the context of use. Answering the questions arising from this analysis leads to the possibility of improvement in design. Indeed, at several points during analysis our inner voice has probably cried out, “There must be a better way.”

One of the first considerations in designing documents is that they have always been (and presumably always will be) hierarchical in their structure—whether they are encoded on clay tablets or in electronic characters. A document can be seen as a set of nested structure of components. This is why models of documents are often expressed as tree diagrams because such a hierarchy is the best way to represent them.
However, the document component model produced by our analysis represents a network, not a hierarchy. It cannot define a document because it has no definite roots, branches, or leaves.

Consider the component model of the procurement scenario we described earlier. If an Order may be placed by a Buyer Party, a Buyer Party may receive an Invoice, and an Invoice may reference an Order, it is unclear whether the Order structure would be a root, branch, or leaf in a document hierarchy. This is because the context of use for the Order structure is not fully defined in the document component model.

Network models such as document component models (and relational databases) are useful for information storage because they are versatile and reusable for many purposes. They encompass a broad context of use. But they aren’t so good for information exchange.

When it comes to exchanging information, we need to enforce a precise context of use that only a hierarchachical structure describes. Put another way, Document Engineering extends conventional data modeling or database schema design practice by also defining the models needed when applications or databases exchange information.

To create a suitable hierarchical model of a document we first select the structural component required as the root of the hierarchy. We call this the entry point. Then we add the required roles and associations as dictated by the business rules and requirements of the document’s context of use. We refer to this task as assembling a document model.

For example in the Order, Buyer Party, and Invoice scenario, the document assembly model for an Order document may implement the rule that “an Order requires a Buyer Party.” The document assembly model for a Sales Report might use another rule that “a Buyer Party may have one or more Invoices covering one or more Orders.” And the document assembly model for an Invoice may contain the rule “an Invoice for a Buyer Party may relate to one or more Orders.” In each case the assembly of components is
driven by rules based on the specific context of use. This example highlights the fact that certain components may be reused when creating different document assembly models for different contexts.

### 7.10 Implementing Models

Any To-Be models for new processes or documents are purely theoretical unless we represent them in a physical form so that they can be used in applications.

So the final phase in the Document Engineering approach is to use our conceptual models for some practical effect. Specifically this means using our document models for validating document instances and using our process models in software applications that control business services.

---

**Business applications exist to enforce some set of rules or constraints about information or process.**

The reason business applications exist is to enforce some set of rules or constraints about information or process. So any application can be thought of as a software artifact that presents, collects, and manipulates information according to these rules.\(^\text{12}\)

The fact that our models represent the rules and requirements of a context of use means that any software application that satisfies those requirements should be able to rely on these models to determine their behavior. In doing so, the rules about information and process captured by the models remain explicit or externalized from the software that enforces them. This implementation is preferable to any that has these rules buried in the application logic where they are not easily examined or modified.

To *realize* model based applications we need to create physical, computable artifacts from our models. While none of the analysis and design methods we’ve discussed so far have anything inherently to do with XML (or any other syntax), the best available way to realize physical models from our conceptual ones is to encode them in an XML schema language.

These activities are discussed in Chapter 15, “Implementing Models in Applications.”
7.10.1 Encoding Document Implementation Models

Document assembly models are realized by encoding them as what we call document implementation models.

This is actually the inverse of the harvesting task in which we took document artifacts and removed their implementation features to yield the underlying semantics. Now we can apply new implementation features in a more consistent and formal way.

With XML encoding we can choose from any of several different XML schema languages we discussed in Chapter 2. Each offers different tradeoffs in simplicity, expressive power, and maintainability.

Along with the decision about XML schema languages comes the potential to reuse patterns from existing XML vocabularies. Some XML vocabularies (such as UBL) provide re-usable definitions for common components such as Item, Party, Code, Address, Amount, and Location. These are usually published as physical models or schemas using one language as their authoritative format.

Choice of schema language alone is not sufficient to encode a document implementation model. Regardless of the language chosen, it is also necessary to develop or adopt grammatical rules that govern the techniques for encoding that language.

Business service interfaces can then validate document exchanges against these schemas and process the content as required.

7.10.2 Encoding Business Process Implementation Models

Business process implementation models encode the To-Be process, collaboration, and transaction models we defined together with any patterns we have adopted or adapted for our new designs. As we noted in Chapter 3, there are many metamodels for defining business processes, so we need to select a metamodel appropriate for our implementation model.
Because XML permits extensible vocabularies, it is increasingly common to find business process metamodels expressed as XML schemas (such as the ebXML BPSS or BPEL). We encode implementation models as instances of XML documents based on the schemas these metamodels describe.

Business service interfaces can then interpret these documents to guide the processing of the documents they receive.

### 7.11 Summary of Modeling Phases and Artifacts

Figure 7-3 summarizes the modeling artifacts that are developed by following the Document Engineering approach. These artifacts are designed to ensure an effective and sufficient transition of information between the various phases.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzing the Context</td>
<td>UML use case diagrams</td>
</tr>
<tr>
<td>Analyzing/Designing Business</td>
<td>Business Domain View Worksheet</td>
</tr>
<tr>
<td>Processes</td>
<td>UML use case diagrams</td>
</tr>
<tr>
<td>Analyzing/Designing Business</td>
<td>Business Process Area Worksheet</td>
</tr>
<tr>
<td>Collaborations</td>
<td>UML activity diagrams</td>
</tr>
<tr>
<td>Analyzing/Designing Business</td>
<td>Business Transaction View Worksheet</td>
</tr>
<tr>
<td>Transactions</td>
<td>UML sequence diagrams</td>
</tr>
<tr>
<td>Applying Patterns to Business</td>
<td>Document checklist</td>
</tr>
<tr>
<td>Processes</td>
<td>Document inventory</td>
</tr>
<tr>
<td>Analyzing Documents</td>
<td></td>
</tr>
<tr>
<td>Analyzing Document Components</td>
<td>Consolidated table of content components</td>
</tr>
<tr>
<td>Assembling Document Components</td>
<td>UML class diagram</td>
</tr>
<tr>
<td>Assembling Document Models</td>
<td>UML class diagram or spreadsheet</td>
</tr>
</tbody>
</table>
### Table

<table>
<thead>
<tr>
<th>Assembly Model</th>
<th>XML Schema for Document Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementing Model-Based Applications</td>
<td>XML Schema for Process Models</td>
</tr>
</tbody>
</table>

### Figure 7-3 Summary of Modeling Phases and Artifacts

### 7.12 Key Points in Chapter 7

- Document Engineering is a coherent and pragmatic approach for modeling documents and services that provide solutions that are practical and effective.
- There is no single correct way to create document and process models.
- This approach embodies the principle of loose coupling through document exchanges.
- Document Engineering exploits different analysis techniques for reaching the same goal.
- Choosing a pattern for a business process entails adopting a predefined context of use.
- Effective design involves the analysis, reuse, and creation of patterns.
- We exploit conceptual models to bridge the gap between knowing what to do and actually doing it.
- Requirements are constraints on possible solutions that must be satisfied for the solution to be considered acceptable.
- Collecting requirements and rules is a heuristic and iterative exercise.
- We use the context of use to organize and analyze requirements and rules.
- Businesses exist to exploit the results of their private processes in their public ones.
• This tension between public and private processes induces businesses to reuse existing models.

• The richer the document inventory, the more effective any analysis will be.

• Presentational structures are often the most salient patterns in narrative documents.

• Naming components is a contentious, iterative and ongoing activity.

• More rigorous techniques for assembling structural components produce more predictable results.

• Document Engineering extends conventional data modeling by also defining the models needed to exchange information.

• Business applications exist to enforce some set of rules or constraints about information or process.

• The implementation language influences the potential to reuse existing patterns.

7.13 Notes

1. UN Economic Commission for Europe, *UN/CEFACT Modeling Methodology, version 8.1* (CEFACT/TMWG/N090R8.1, 2001), Chapter 8. The UMM is a set of metamodels and a prescriptive methodology for using them in the design of business processes and their associated documents.


3. The BPSS is a metamodel described as an XML schema that can be used to define public business process models as XML documents (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-bp).

4. Defining information components as objects rather than data allows the behavior of the component to be attached to its definition. While this has relevance to building reusable programming functions, it is less applicable to loosely coupled document modeling, so we shall focus on analysis of the static component rather than its behavior.
5. The artifacts created by these phases roughly correspond to the business requirements view (BRV) and business transaction view (BTV) in the UMM.


7. The source of the correct quote is *Devotions upon Emergent Occasions, No. 17* (1624).


9. Any component with a number also has a text title, but it would be a stretch to treat each text title as a label of the type of content.

10. Normalization techniques are taught in almost every database book and course. We recommend the classic text by Chris Date, *An Introduction to Database Systems Volume 1* (Addison Wesley, 1982).

11. In a complex problem context, the number of possible associations can become unmanageably large, so in practice we need to focus on the most important associations. We might need to simplify the pattern or model somewhat; for example, the most important associations in an organizational model would be the hierarchical reporting relationships, and while some “dotted-line” responsibilities might exist, we might safely ignore them in some contexts.