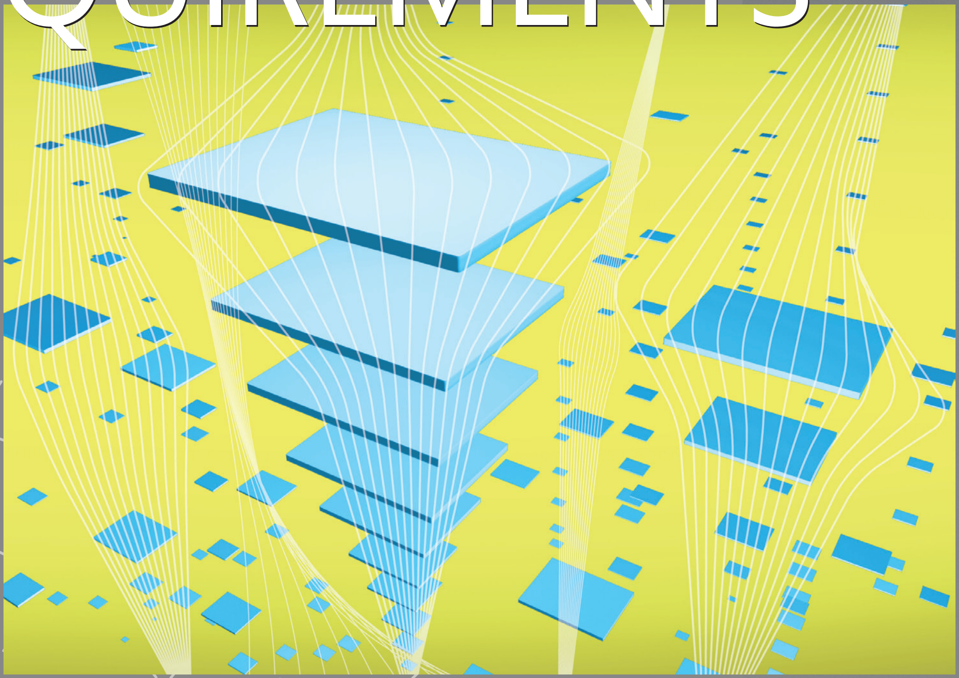
Microsoft

BEST PRACTICES

VISUAL MODELS FOR SOFTWARE REQUIREMENTS



Joy Beatty and Anthony Chen

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2012939549
ISBN: 978-0-7356-6772-3

Printed and bound in the United States of America.

First Printing

Acquisitions Editor: Devon Musgrave
Developmental Editor: Devon Musgrave
Project Editor: Carol Dillingham
Editorial Production: Online Training Solutions, Inc.
Copyeditor: Kathy Krause
Indexer: Jan Bednarczuk
Cover Illustration: John Hersey

We dedicate this book to all the unrecognized analysts who don't get the credit they deserve for making their projects successful.

Contents at a Glance

Contents

Foreword

The most striking thing about requirements work is the enormous difference between what academics think it involves and what people in industry actually do.

The academics think they are far ahead, because they have a wide range of models and techniques, complete with experimental studies (done with specially tamed industrial tribespeople), theoretical analyses, and enormous textbooks full of excellent advice. They can't see why people in industry are being so slow to adopt their methods.

The people in industry think they are far ahead, because they have years of experience, software that works (after a bit of pushing and shoving), and proven methods of managing requirements with traceability matrices, reviews, configuration management, and attributes for priority and status. They can't see why people in academia are being so slow to catch up with reality.

It's like watching two cyclists on a circular racetrack, 180 degrees apart, endlessly circling.

That's why it is so good to see this book from Joy Beatty and Anthony Chen. They are practitioners who speak from their own experience. But—and this is the crucial thing—they are familiar with the range of models advocated by researchers, and even better, they have steadily incorporated more and more of these into their practice. Now they have reached the point where they can see that the models they are using enable them conveniently and effectively to analyse all the requirements they come across. They've seen and heard the academics talking about, say, goal modeling using KAOS or i*. They've seen challenged projects that only needed a context model to inject clarity, or the disaster that looms on projects that lack something as simple and traditional as a data dictionary. And they have a practical handle on the essential fact that you have to use all these things together.

They've arrived at a clear understanding that in a requirements process, as in any system or product, the whole is more than the sum of its parts. An airframe, a pair of powerful engines, an avionics system, and an aircrew do not make an aircraft until they are integrated. When they work together, something new emerges that none of the parts could achieve on their own: the ability to fly.

To make a requirements process "fly," the first step is to understand that there is more than one kind of requirements model. A shopping list of requirements is invaluable in a contract, but on its own, it's desperately difficult to check for correctness and completeness, and it doesn't offer any suggestions on how to discover requirements, either. Different requirements models are needed to assist with discovering, checking, and analyzing the requirements. The "shopping list" is an output, not the one-and-only input.

Joy and Anthony identify four major classes of requirements model: those dealing with objectives, people, systems, and data.

Their *objectives* come closest to traditional requirements, but starting at a much earlier and more tentative stage, looking at what a business's objectives are and from there working out how those needs can be met.

People, obviously, means looking at who has an interest in the system under design, how they will use it, and what they want from it.

Systems means exploring the context, interfaces, and events that govern what the new system will have to do. This is largely a traditional set of analysis techniques, often considered outmoded by those subject to the whims of software fashion, and it is creditably brave of Joy and Anthony to face up to this and to state clearly that old—even if incomplete—does not mean wrong. The point is, of course, that 1970s-style system analysis on its own was not enough—for example, it often failed for lack of proper attention to *objectives*.

Finally, *data* means defining the information that is needed by business users and exploring how it is used within the system. Again, much of this is very traditional, though it covers not only data analysis but state models and report analysis—a modern take on an old topic.

There is a necessary complexity here. Requirements models interlock. Objectives relate to features, which relate to processes, which relate to use cases, which relate to the user interface. Joy and Anthony show how this requirements architecture—you could call it a meta-model—can be tailored to the individual project. They have tried it, over and over, and it works.

The approach in this book is designed for software that supports business processes. Related but distinctively different requirements processes are needed for other kinds of projects, such as developing a family of mass-market products that include both hardware and software. Joy and Anthony focus specifically on one world: the world of software for businesses. The result is an innovative but compelling requirements approach.

- Ian Alexander, April 2012

Introduction

Visual requirements models are one of the most effective ways to identify software requirements. They help the analyst to ensure that all stakeholders—including subject matter experts, business stakeholders, executives, and technical teams—understand the proposed solution. Visualization keeps stakeholders interested and engaged, which is key to finding gaps in the requirements. Most importantly, visualization creates a picture of the solution that helps stakeholders understand what the solution will and will not deliver. Despite this fact, many business analysts and product managers continue to create nonvisual requirements using spreadsheets or documents listing thousands of line items. These unwieldy documents are overwhelming, boring to review, and extremely difficult to analyze for missing requirements. Such practices are a symptom of the state of current requirements training, which is often focused on how to write a good requirement rather than how to analyze an entire solution.

This book will help business analysts, product managers, and others in their organizations use visual models to elicit, model, and understand requirements. It describes a simple but comprehensive language of visual models for software requirements called RML (Requirements Modeling Language) that is a collection of best-practice models that have commonly been used in industry in an ad-hoc fashion.

Who Should Read This Book

Although this book is geared primarily toward business analysts and product managers, we think that project managers, developers, architects, and testers will get a tremendous amount of value out of the book because it can help them understand the standard of information that they should be receiving to make their jobs easier. Throughout the book, we commonly refer to the person doing the work as "the analyst," because this role has many different titles across organizations. When we refer to "you," we are also referring to "the analyst."

We want to be up front and mention that our experience has primarily been with projects that are geared toward building software that operates within an existing infrastructure, such as internally facing information technology (IT) systems, large-scale consumer-facing software as a service (SaaS) systems, and cloud systems. Although we have used RML on stand-alone ("packaged") software and embedded systems, those types of projects have not been our primary focus. However, based on our limited experience with these systems, we still think that readers working with those systems will find incredible value in RML, and we look forward to receiving feedback from those readers to improve it.

Assumptions

This book does not cover basic information on requirements; therefore, it assumes that you have existing foundational knowledge about how to write software requirements. It expects that you have a basic understanding of software development processes such as iterative, waterfall, and agile methods, and how requirements fit into those approaches.

Who Should Not Read This Book

If you are just starting out as a business analyst, you should probably read *Software Requirements* by Karl Wiegers (Microsoft Press, 2003) before reading this book, for an overview of requirements practices. If you are developing shrink-wrapped consumer software, some of the concepts will be useful, but you might find the business orientation distracting. If you are a product manager who focuses on the strategy or marketing aspect of software products rather than on software construction, then this book might not be a good fit, because it heavily emphasizes how to design features for high end-user adoption and satisfaction.

Organization of This Book

We have organized this book so that you can use it as a reference guide.

Part I, "An Introduction to Models," introduces models in general and then goes on to discuss RML and the four classifications of models: objectives models, people models, systems models, and data models (OPSD).

Each chapter in Parts II through V covers one RML model and has a consistent layout, including:

- A story that relates the model to the real world.
- A definition of the model.
- The model template.
- A suggestion of which tools to use to create the model.
- A fictional example.
- Explanations of how to create and use the model.
- An exercise so that you can practice using the model.

The exercise in each chapter is in the context of one sample project that is used throughout all chapters.

Part VI, "Models in the Big Picture," explains how to select the models and how to use models together to derive requirements.

Appendix A contains two quick lookup models grids as references for how to select models. Appendix B suggests general guidelines for creating models, including metadata for all models and template tips. Appendix C contains the answers to all of the exercises in the book. There is also a Glossary defining the terms that are used throughout the book.

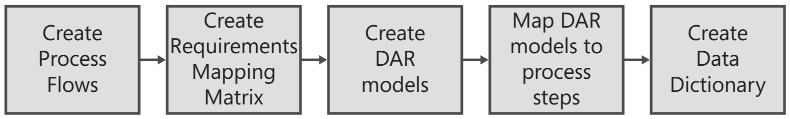
Finding Your Best Starting Point in This Book

You can read the book straight through, but for some people, reading Part VI first might help create context before you delve into the details of each model. The following table provides more guidance.

If you are	Follow these steps
New to requirements modeling or visual modeling in general	Read the book from front to back so that you can get an introduction to requirements models, learn about the individual models, and finally put them all together.
Familiar with visual requirements modeling and are a business analyst who uses similar models already	We suggest that you look at all of the chapters to understand how RML treats the visual models differently than other modeling languages. However, you might find Part VI more useful to start with for understanding the more advanced topic of how to select models and use them together on projects. You can then refer to the specific model chapters as you need them on your project.

Models Quick Start

This book contains a tremendous amount of information to absorb about models. The prospect can be overwhelming, so we have developed a way for you to get started with models that uses as few models as possible but still creates significant value for projects. This quick-start method fits most IT-based projects. The following Process Flow provides an overview of this approach.


Create Process Flows
Create Requirements Mapping Matrix
Create DAR models
Map DAR models to process steps
Create Data Dictionary

As shown in the diagram, you start by creating the Process Flows. Next, you create a Requirements Mapping Matrix (RMM) based on the Process Flow steps. Then you create Display-Action-Response (DAR) models for screens and map them against business processes. Finally, you create Data Dictionaries to ensure that all fields are covered and that the validation rules are known.

This leaves out a lot of the value of the other models, but it is a series of steps that can be adopted without major upheaval. The result is that your requirements will be organized by process steps and your screens will also be mapped to process steps to ensure that the key processes are satisfied by the user interface.

Conventions and Features in This Book

This book presents information by using conventions designed to make the information readable and easy to follow.

- Every chapter starts with a non-software story in italics to set context for the reader.
- All RML model names are capitalized throughout the book. Models from other modeling languages that are not part of RML are not capitalized.
- The building blocks of RML models are called elements, and those model elements are not capitalized so that they are not confused with model names.
- The glossary at the end of this book contains terms that we consider to be important terms for RML. These terms appear in italics throughout the book.
- Each model template includes a Tool Tip reader aid that provides suggestions for which tools to use to create that model.

Companion Content

You are welcome to download the RML model templates to use as you create the models from this book on your projects. A full set of templates for the RML models is available at:

http://www.microsoftpressstore.com/title/9780735667723

Instructions in the compressed file explain how to use the templates. A brief overview is repeated here: Download the compressed file and extract its files to a convenient location. There is one template for each model. The models that are Microsoft Visio files include both a template and a stencils file, both of which are required to make the template work correctly. The rest of the templates are either Microsoft Excel or Microsoft Word formats. The quick lookup models grids are also in the compressed file.

Acknowledgments

From our team at Seilevel, to our requirements colleagues around the world, to our customers who inspired and helped us refine RML over the years, this book would not have been possible without the collaboration of all of you.

Many thanks to the employees at Seilevel who helped with research, reviews, writing, editing, drafting models, and asking really good and hard questions: Joyce Grapes, James Hulgan, Betsy Stockdale, Michael Liu, Candase Hokanson, Jeremy Gorr, Balaji Vijayan, Marc Talbot, Matt Offers, Ajay Badri, Jason Benfield, Geraldine Mongold, Kell Condon, Clint Graham, David Reinhardt, Weston Eidson, Abdel Mather, Kristin DiCenso, Rob Sparks, and Lori Witzel.

Our deepest gratitude goes to the many reviewers who took time to read the manuscript and give their thoughts and critiques to help improve the book: Joyce Statz, Kent McDonald, Sarah Gregory, Ljerka Beus-Dukic, Mary Gerush, Karl Wiegers, Ellen Gottesdiener, Scott Sehlhorst, Ivy Hooks, and Anne Hartley. We want to extend a special thank you to Karl Wiegers and Ian Alexander, both of whom offered authoring mentorship and acted as philosophical sounding boards on the models.

We are deeply appreciative of the hardworking and fun editorial team who made this book a reality. Thank you to our acquisitions and developmental editor, Devon Musgrave, and our project editor, Carol Dillingham, both at Microsoft Press. We also want to thank our project manager and copyeditor, Kathy Krause; desktop publisher, Jean Trenary; proofreader, Jaime Odell; graphic artist, Jeanne Craver; and indexer, Jan Bednarczuk.

Finally, we want to thank our families, who endured the long writing process with us. Joy is grateful to her husband, Tony Hamilton, who helped her keep her sense of humor throughout the process, and her daughter, Skye, who was born in the middle of writing this book and learned to sleep through the night just as we finished writing. As it turns out, writing this book was a lot like having a baby: months of incubation, preparation, and nurturing. Anthony is appreciative of his wife, Gloria, for her support, and of his daughter, Mason, for letting him work when playing sounded like more fun and for being very quiet during conference calls. Finally, Anthony wants to thank Joy. Without her to drive the writing of this book through her sheer force of will, it never would have happened.

Errata & Book Support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://www.microsoftpressstore.com/title/9780735667723

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We Want to Hear from You

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in Touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

CHAPTER 16

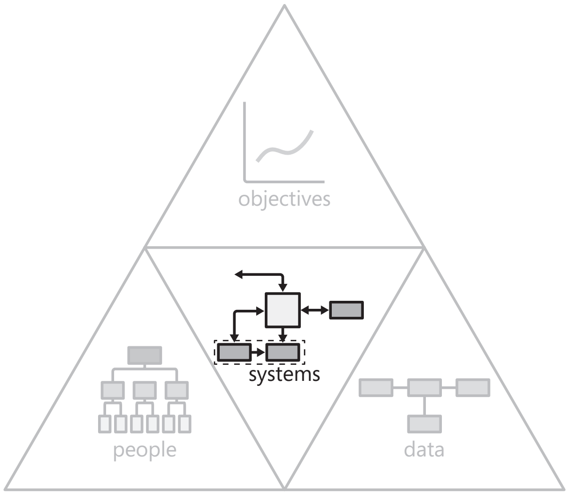
Decision Table

The other day, I was trying to teach my friend how to play Texas Hold 'em poker. In the game, there are only a few actions that you can take: call, raise, fold, or check; yet the game is very complex, and it isn't at all obvious when you should take one of those actions.

I realized that there were just a few factors to consider, but that they created too many combinations to make me feel sure that I had covered all the important situations for my friend. Some of the factors going into the decision about your action are which seat you are in, how many other people have stayed in, how many have yet to act after you, the probability of improving your hand compared to the current pot size, and the type of each of the players.

For example, if the size of the pot compared to how much I have to put in is too small compared to my chances of improving my hand, then I usually fold regardless of any other factors. However, if everyone before me has folded and there are only one or two people after me, I might raise, unless I know that the players yet to act are the type to protect their antes (blinds). On the other hand, even if I have a good hand with good chances for improving it, if there are one or two really aggressive players still in, I might fold just to stay out of their way.

These kinds of multifactor decisions are very common in software development. The Decision Table is an RML systems model that helps you analyze all the permutations of complex logic in a comprehensive way. Decision Tables are used to answer the question, "Under what conditions will this outcome occur?" or "Given these conditions, what outcome should I choose?" The format of the model allows you to easily ensure that all possible conditions are being checked and acted upon properly. Decision Tables are used if there is no specific order to evaluating the decisions. If the decisions need to be made in any kind of order, a Decision Tree should be used instead (see Chapter 17, "Decision Tree").

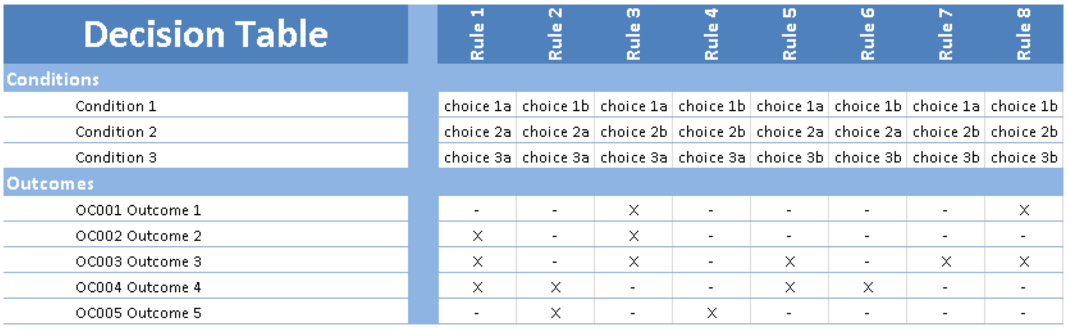


Decision Tables show all possible combinations of a set of conditions and their corresponding outcomes, represented in a grid. The Decision Table format allows you to ensure completeness. Because the full quantity of combinations is known, you can be 100 percent certain that you have looked at every permutation. It is easier for technical teams to use Decision Tables than tree structures because every option is shown in an organized format. A Decision Tree can get cumbersome when it attempts to show all possible conditions and outcomes.

The advantages of a Decision Table over a Decision Tree are visual traceability and speed. You can instantly trace an outcome back to the conditions that cause that particular outcome. You can also always trace all the potential outcomes from a condition. A table is faster to create than a Decision Tree, because with Decision Trees you have to rearrange your branches for every new branch that you add. If you decide to create both, it usually is best to create a Decision Table first, before you create your tree, so that you have an idea of what the layout should look like.

Decision Table Template

A Decision Table is represented as a grid, as shown in Figure 16-1. The top row contains labels for the business rules. The first column contains all the possible conditions and outcomes. Each of the remaining columns in the grid represents the outcomes valid for the specified choices. Collectively, the combination of choices and outcomes in a column make up each business rule.



Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Conditions								
Condition 1	choice 1a	choice 1b	choice 1a	choice 1b	choice 1a	choice 1b	choice 1a	choice 1b
Condition 2	choice 2a	choice 2a	choice 2b	choice 2b	choice 2a	choice 2a	choice 2b	choice 2b
Condition 3	choice 3a	choice 3a	choice 3a	choice 3a	choice 3b	choice 3b	choice 3b	choice 3b
Outcomes								
OC001 Outcome 1	-	-	X	-	-	-	-	X
OC002 Outcome 2	X	-	X	-	-	-	-	-
OC003 Outcome 3	X	-	X	-	X	-	X	X
OC004 Outcome 4	X	X	-	-	X	X	-	-
OC005 Outcome 5	-	X	-	X	-	-	-	-

FIGURE 16-1 The Decision Table template.

A condition is an individual check such as "Lives in the United States - Yes or No" or "Marital status - single, married, divorced, or widowed." We use the term "condition" instead of the more common term "decision" because "decision" implies that there is an order to the decision process, whereas "condition" does not.

The choices for a condition are the set of possible values for that condition. Choices can be binary, such as Yes or No or True or False; they can be multi-valued, such as ages 0-9, 10-21, 22-34, and 35+; or they can be a dash (-), meaning that the choice is irrelevant. The outcomes can have unique identifiers on them to help reference them from requirements.

Each outcome cell either contains an X, a number, a dash, or is left blank, as explained in Table 16-1.

TABLE 16-1 Outcome and Choice Intersection Elements

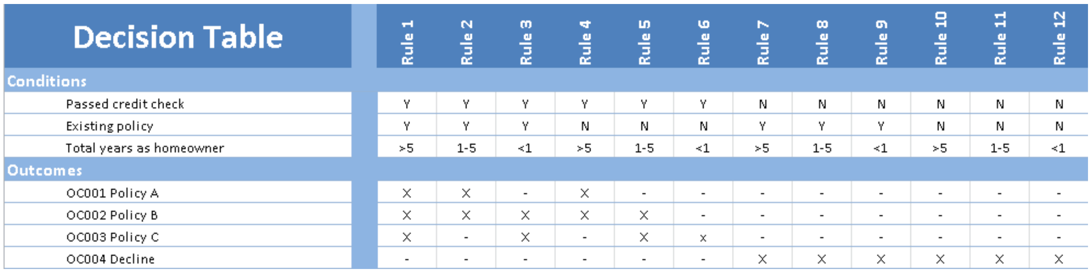
Element	Meaning
X	Outcome applies when the choices are valid
Number	Outcome applies when the choices are valid; the outcomes should be executed in a specified order
-	Outcome is irrelevant (does not apply) when the choices are valid
Blank	Outcome is unknown; follow-up is needed

A rule is the particular set of choices for each condition that have to match for the outcomes to apply. For example, the condition "lives in the United States = yes" and the condition "Marital status = single" could form the choices that make up a rule that says a specific outcome applies. The rules don't have names; they are just permutations of all the possible choices of all the conditions and valid outcomes. Rules eventually become business rules in your requirements. For example, in the template, if the choices choice 1a, choice 2a, and choice 3a are true, then outcomes 2, 3, and 4 apply.

Tool Tip Decision Tables are typically created as Microsoft Excel or Microsoft Word tables or in a requirements management tool that allows you to create tables of requirements.

Example

An insurance company has a formalized process for determining which homeowner policies a customer is eligible for. Their decision process is modeled in the Decision Table in Figure 16-2. The model provides the assurance that all possible combinations are considered, because every combination of choice for every condition is included.

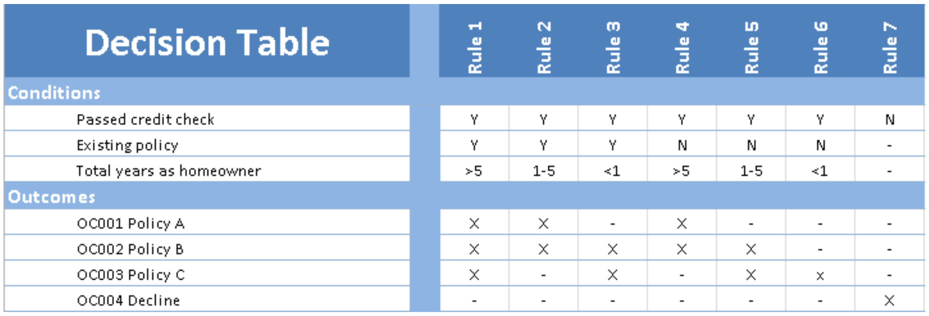


Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
Conditions												
Passed credit check	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Existing policy	Y	Y	Y	N	N	N	Y	Y	Y	N	N	N
Total years as homeowner	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1
Outcomes												
OC001 Policy A	X	X	-	X	-	-	-	-	-	-	-	-
OC002 Policy B	X	X	X	X	X	-	-	-	-	-	-	-
OC003 Policy C	X	-	X	-	X	x	-	-	-	-	-	-
OC004 Decline	-	-	-	-	-	-	X	X	X	X	X	X

FIGURE 16-2 The Decision Table for insurance policies.

For example, the Decision Table tells us that if a customer has passed a credit check, has an existing policy, and has been a homeowner for 20 years, then he is eligible for policies A, B, and C. If the same customer did not have an existing policy, then he would be eligible for policies A and B.

Looking at the complete table, you can see where unnecessary columns can be pruned. For example, being a homeowner and having existing policies with the company are irrelevant if the person fails the credit check. Figure 16-3 shows the simplified table.



Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7
Conditions							
Passed credit check	Y	Y	Y	Y	Y	Y	N
Existing policy	Y	Y	Y	N	N	N	-
Total years as homeowner	>5	1-5	<1	>5	1-5	<1	-
Outcomes							
OC001 Policy A	X	X	-	X	-	-	-
OC002 Policy B	X	X	X	X	X	-	-
OC003 Policy C	X	-	X	-	X	x	-
OC004 Decline	-	-	-	-	-	-	X

FIGURE 16-3 The simplified Decision Table for insurance policies.

Creating Decision Tables

The high-level steps for creating a Decision Table are outlined in Figure 16-4. We suggest an order to the steps; however, you might do this differently, as discussed later in this chapter, in the "Identify Outcomes" section. To illustrate how to create Decision Tables, this section continues with the example scenario from the previous section.

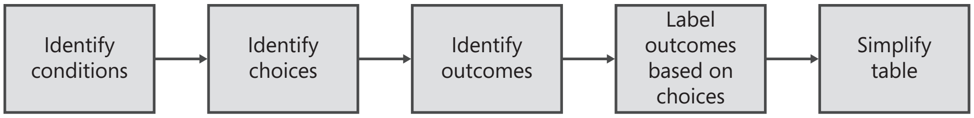
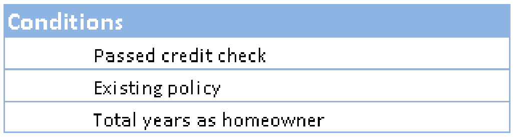


FIGURE 16-4 The process for creating a Decision Table.

Identify Conditions

Think about all the potential conditions that apply to the situation and list them in the first column of your table. Conditions can be decisions that people make, data attributes that trigger various business rules, or any other factor (Gottesdiener 2005). Each condition needs its own row in the table. If you combined multiple conditions, it would be very hard to check for completeness. Figure 16-5 shows the first part of the table with just the possible conditions. A condition will often reference specific data fields from a Data Dictionary. In those cases, use the <*object.field*> notation, as described in Chapter 21, "Data Dictionary."



Conditions
Passed credit check
Existing policy
Total years as homeowner

FIGURE 16-5 The conditions.

Identify Choices

When you identify choices, you first need to determine what the choices are, then you might re-order your conditions in the table based on the choices, and then you enumerate the choices to build the rules columns.

Determining Valid Choices

After you have the conditions, consider the possible choices for each condition. Look at your Data Dictionary to identify valid values for the field. Some conditions will have binary choices of Yes and No or True and False, but the choices can be more complex, such as a range of numbers or matching words. There is no requirement that the conditions all be binary; in fact, the table can be simpler if a single multi-choice condition is used instead of several binary conditions. For example, if the condition is "Years as a homeowner," you might have three potential choices for that condition that should be modeled: more than five years, one to five years, and less than one year.

Make sure that the choices you use reflect all possible choices for the condition. If you miss any possible choices, your table will be incomplete and you cannot guarantee that you have all of the requirements. For a particular condition, the choices must also be mutually exclusive, in that only one can exist at a time. A condition cannot have overlapping ranges such that each could simultaneously match because they share values. For example, the ranges 0–1 and 1–5 overlap because each includes the number 1. If you need to specify ranges like this, use the following type of range definitions instead: <1 and 1–5, or <=1 and >1–5.

Reordering Conditions

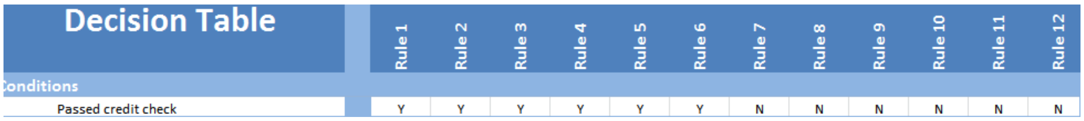
You do not need to list the conditions in any specific order; however, it is generally easier to make a table when they are listed in the order of number of choices. The condition that has the fewest choices should be the first condition. For example, if you have a rule such as, “If the customer failed the credit check, then automatically decline him,” then you can cut out half of your permutations. You don’t need to do any work to evaluate the rest of the conditions; they just don’t matter. The condition that has the most choices should be the last condition listed. This order puts the most constraining conditions at the top of the list, which aids you in reviewing and simplifying the table.

Enumerating Choices

Now you add rules (columns) to enumerate all possible combinations of choices for all conditions. If you multiply the number of choices together for each condition, you can determine how many columns you need. If you have just one condition with two choices, you will need only two columns. If you have three conditions with two choices each, you will need eight columns (2 x 2 x 2 = 8). In this example, there are two choices for the first condition (Y or N), two for the second condition (Y or N), and three for the third condition (>5, 1–5, or<1), so there are 12 columns (2 x 2 x 3 = 12).

In addition to each choice being mutually exclusive, each rule needs to be mutually exclusive. This means that you should have no columns with the same set of choices.

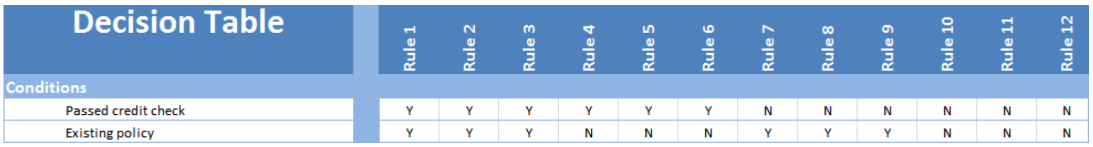
The order of the rules should make it easy to recognize that all possible choices of conditions are being examined. In the first row, group all the like choices together. In the example, for the first condition, half will be Y and half will be N, so make the first four columns Y and the second four N, as in Figure 16-6.



Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
onditions												
Passed credit check	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N

FIGURE 16-6 The first condition with choices.

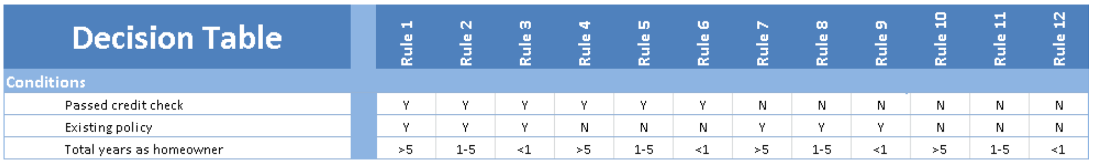
For the second condition, half of the Ys from the first condition will be Y and half will be N. This results in the pattern in Figure 16-7.



Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
Conditions												
Passed credit check	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Existing policy	Y	Y	Y	N	N	N	Y	Y	Y	N	N	N

FIGURE 16-7 The second condition with choices.

Finally, in the third condition, alternate the three choices again to get the pattern in Figure 16-8. When you complete the conditions and choices of a Decision Table, you will get a result that looks like this. Remember that you will have more columns if you have more conditions or valid choices for conditions.

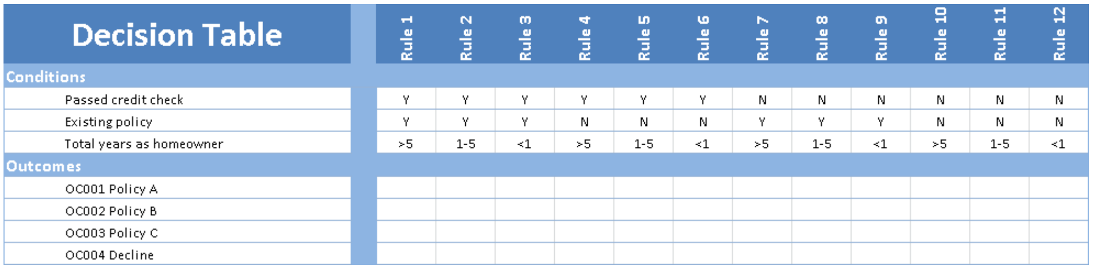


Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
Conditions												
Passed credit check	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Existing policy	Y	Y	Y	N	N	N	Y	Y	Y	N	N	N
Total years as homeowner	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1

FIGURE 16-8 All conditions with all choices enumerated.

Identify Outcomes

The combinations of choices lead to one or more outcomes. Outcomes are the decisions, conclusions, or actions that occur when the choices are valid. Add the outcomes to the first column of your table, below all of the conditions. Figure 16-9 shows our same scenario with outcomes added.



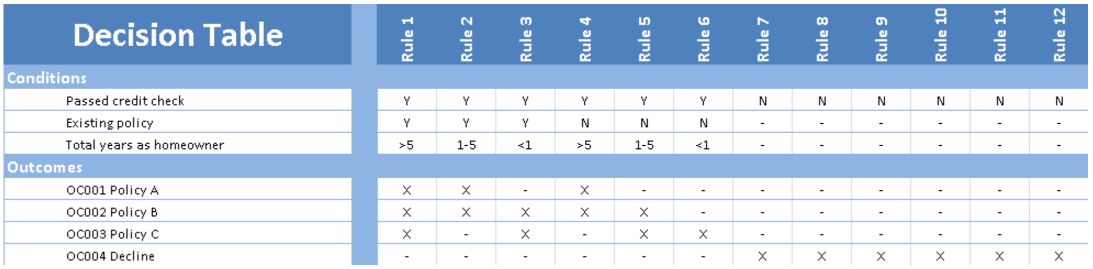
Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
Conditions												
Passed credit check	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Existing policy	Y	Y	Y	N	N	N	Y	Y	Y	N	N	N
Total years as homeowner	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1
Outcomes												
OC001 Policy A												
OC002 Policy B												
OC003 Policy C												
OC004 Decline												

FIGURE 16-9 Outcomes.

You might find it easier to identify the outcomes first. If you know your outcomes and want to determine the conditions under which they are valid, start with the outcomes part of the table and then identify conditions and choices. It is possible that you might start with a few business rules, so creating the table with just a few rules filled in can quickly show where there are gaps in your information.

Label Valid Outcomes by Choice Combinations

Each column now represents a possible combination of choices for the conditions. Label the valid outcomes under the combination of choices. If you know that an outcome is not valid, mark it with a dash, and if you are not sure, leave it blank to follow up later. Figure 16-10 shows the resulting Decision Table for the scenario.



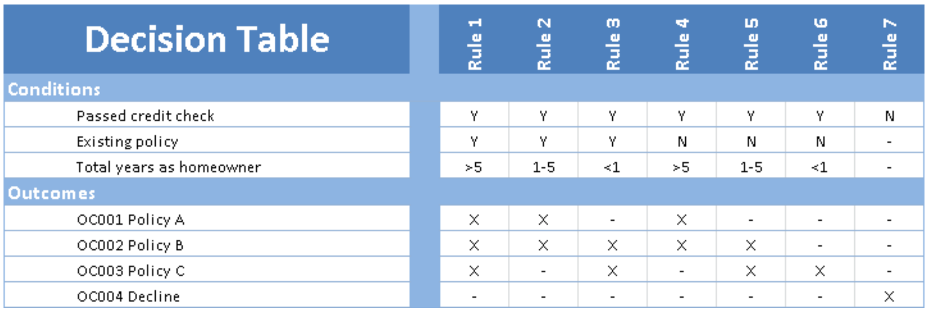
Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
Conditions												
Passed credit check	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Existing policy	Y	Y	Y	N	N	N	-	-	-	-	-	-
Total years as homeowner	>5	1-5	<1	>5	1-5	<1	-	-	-	-	-	-
Outcomes												
OC001 Policy A	X	X	-	X	-	-	-	-	-	-	-	-
OC002 Policy B	X	X	X	X	X	-	-	-	-	-	-	-
OC003 Policy C	X	-	X	-	X	X	-	-	-	-	-	-
OC004 Decline	-	-	-	-	-	-	X	X	X	X	X	X

FIGURE 16-10 The complete Decision Table.

Simplify the Decision Table

Because so many columns are necessary even with just a few conditions, it is best to try to simplify your table as early in the process as feasible. If there are specific rules that are only dependent on a few of the conditions, that means that you do not need to evaluate other conditions. You should combine the rules to remove the ones that differ in conditions and not in outcomes.

When a condition is irrelevant to the outcome, put a dash in the cell for that condition. Do not just haphazardly delete columns, though! You must ensure that all combinations are included in the table, using the dash to represent those conditions whose choices do not matter. For example, you know that when a customer fails the credit check, she is declined, so you do not need to evaluate the existing-policy or years-as-a-homeowner conditions. Figure 16-11 shows the example after it has been simplified.



Decision Table	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7
Conditions							
Passed credit check	Y	Y	Y	Y	Y	Y	N
Existing policy	Y	Y	Y	N	N	N	-
Total years as homeowner	>5	1-5	<1	>5	1-5	<1	-
Outcomes							
OC001 Policy A	X	X	-	X	-	-	-
OC002 Policy B	X	X	X	X	X	-	-
OC003 Policy C	X	-	X	-	X	X	-
OC004 Decline	-	-	-	-	-	-	X

FIGURE 16-11 The simplified Decision Table.

In simplifying your table, you might discover that some conditions are always irrelevant. You can remove those conditions and further condense your table. Finally, if, after you complete your table, you see that certain outcomes are always executed in tandem, you can combine those outcomes into a single outcome.

Using Decision Tables

Decision Tables can make very complex decisions appear orderly and complete because the table communicates a lot of information in a very compact format.

Making Decisions

A business can use a Decision Tree for training users how to make decisions, but a Decision Table is hard to read for these purposes. More commonly, a Decision Table is implemented in the system to make the decisions automatically.

Driving Completeness

The value provided by Decision Tables stems from the ability to walk through all possible combinations of choices. Assuming that every outcome, condition, and possible choice is identified, you can literally see that all the potential choices have been considered, and therefore your requirements for the decision scenario can be considered complete.

When you are analyzing Decision Tables, if you have any cells you do not know about, leave them blank and track them for follow-up.

Using Decision Tables with Decision Trees

Decision Tables are used to identify all possible conditions and choice combinations and their resulting outcomes. You can also use them to eliminate combinations that are not relevant. Finally, you can use the Decision Table to build a corresponding Decision Tree to better visualize the decisions and look for additional opportunities to simplify the logic. Generally, if you plan to create an ordered Decision Tree, you should order the conditions in the Decision Table by using the same order that you use to make the decisions in the ordered Decision Tree.

Deriving Requirements

A complete Decision Table indicates that you have modeled all of the requirements and business rules related to a particular set of decisions. You might also need to write out individual statements from your table for developers and testers to work from. Each complete column of the Decision Table represents one business rule you need to write; the rule describes the conditions under which the outcomes occur. Decision Tables can sometimes act as stand-alone requirements for developers and testers.

When to Use

Decision Tables are best when the order of the decisions does not matter. If the order does matter, you can use Decision Tables to identify all combinations of conditions and simplify them before you create a Decision Tree, but you still must create an ordered Decision Tree.

If you are in an elicitation session and have no time to prepare in advance, use a Decision Table with your business stakeholders to elicit an initial draft of the logic, because you can create Decision Tables very quickly.

Modeling Complex Logic

Decision Tables are usually supplements to System Flows (see Chapter 13, “System Flow”), UI Flows (see Chapter 14, “User Interface Flow”), Process Flows (see Chapter 9, “Process Flow”), or Use Cases (see Chapter 10, “Use Case”). The Decision Table model is used to simplify any of these models by removing the complex decision logic from the main flow. This allows the audience to focus on the overall picture of the flow in those models without getting lost in the details.

It is very useful to use Decision Tables alongside these other models when there are validation steps. For example, in a Process Flow to create an order, there might be complex logic to check that the proper fields are completed and formatted correctly, with different error paths if they are not. These validation steps can be represented in a Process Flow, but if there are many of them, it would be easier to read both the Process Flow and the logic steps by putting the validation into a Decision Table. Instead of putting them all in the Process Flow, the analyst can include one simple decision box in the Process Flow, such as, "Are order fields valid?" and can then reference the Decision Table, where that logic is further analyzed.

A Decision Table should be created any time you have a scenario with a series of nested "if" statements. If you find you need to write two to three "if" statements in a row in a Process Flow or Use Case, you probably should use a Decision Table to model them instead.

Particularly if the order of the decisions is not relevant, a Decision Table is usually the best choice, especially if each combination of choices leads to a different set of outcomes. The visual structure of a Decision Table allows you to quickly see the logic represented by your decisions. A five-by-five table is much easier to understand and review than the 25 separate "if" statements that would have to be written up to describe the information in the table.

When Not to Use

Decision Tables are not ideal for documenting decisions that cause you to move around in your decision hierarchy. If you need to show any order to your decision-making process, you simply cannot use a Decision Table as your only model, because these models do not show order. Similarly, you cannot use a Decision Table if you want to show any loops in your logic. You can use a Decision Table to do your initial analysis to prune the tree, and then follow up with the Decision Tree to show the order of the decisions.

Common Mistakes

The following represent the most common mistakes we have seen with Decision Tables.

Missing Permutations

Make sure you calculate the number of possible rules to ensure that you gather them all.

Overlapping Choice Ranges

Ensure that your boundary conditions for ranges don't overlap; be explicit by using >=, <=, >, and < symbols.

Not Combining Rules

Decision Tables can easily become large and unwieldy if you don't reduce the total number of rules. Make sure that you have identified any conditions that are not important for a rule, and then remove the rules that vary by that condition and not by outcome.

Modeling a Sequence of Decisions

Decision Tables should not be used if you are trying to model a sequence of decisions that occur in order; a Decision Tree is more appropriate.

Related Models

Decision Tables outside of RML sometimes call the conditions *factors* and the outcomes *actions*. Most Decision Table nomenclature allows you to have a blank cell when an outcome does not apply, but we recommend that you do not leave cells blank, because you will not be able to tell whether blank means "not valid" or "I haven't determined whether this is valid."

The following list briefly describes the most important models that influence or are enhanced by Decision Tables. Chapter 26, "Using Models Together," contains a more thorough discussion about all related models.

- **Decision Trees** These are used to visually show the decision logic in a tree structure. Also, the Decision Table can help you to create a Decision Tree.
- **Process Flows, System Flows, Use Cases, and User Interface (UI) Flows** Decision Tables are used to model complex logic found in them.
- **Data Dictionaries** The valid choices for conditions are based on data in the Data Dictionary and should use the <*object.field*> notation.

Exercise

The following exercise is intended to help you to gain a better understanding of how to use this model. The exercise is open ended, and therefore the answer you come up with could be substantially different than the answer that we have provided. There are potentially many correct solutions. The answer provides an explanation of how we arrived at our solution. You will gain the most out of the exercise by attempting to do it yourself before looking at the solution. The answers for the exercises can be found in Appendix C.

Instructions

Create a Decision Table for the following scenario.

Scenario

You are on a project to launch a new eStore to sell flamingos and other lawn decorations. You are capturing the rules for handling payment information. You decide a Decision Table will be the best model to capture the rules the system must implement to handle the different forms of payment. You know that the customer can store credit cards in his profile, but he can also choose to pay with a new credit card, check, or gift card. If he pays with a gift card, he will have to pay any balance with another payment form if there isn't enough money available on the gift card. Use this information and your general knowledge about online payment options to create a Decision Table.

Additional Resources

- *The Software Requirements Memory Jogger* has an overview of Decision Tables with a good example in Section 4.11 (Gottesdiener 2005).
- Wiegers's *Software Requirements* has an example of a Decision Tree and Decision Table (Wiegers 2003).
- Chapter 4 of Davis's *Software Requirements Revision* has a description of Decision Trees and Decision Tables (Davis 1993).

References

- Davis, Alan M. 1993. *Software Requirements: Objects, Functions, & States*. Upper Saddle River, NJ: PTR Prentice Hall.
- Gottesdiener, Ellen. 2005. *The Software Requirements Memory Jogger*. Salem, NH: Goal/QPC.
- Wiegers, Karl E. 2003. *Software Requirements, Second Edition*. Redmond, WA: Microsoft Press.

CHAPTER 25

Selecting Models for a Project

When I decide to work on a project around my house, I start by selecting the tools to use. I first consider what type of project I'm doing and then consider what step in the overall project I'm working on, so that I can pick the right tool. For example, if I'm cooking, I use tools such as a mixer, measuring spoons, a rolling pin, a frying pan, and a spatula. I can further narrow my tool selection by the type of cooking I'm doing. If I'm baking, I would more likely use a mixer than a frying pan. Finally, I pick the tools I need based on which steps in my cooking project I'm executing. Initially, I use measuring spoons and cups, then I use a mixer, then I might use a spatula or rolling pin, and finally I might use a baking dish and cooling rack near the end.

On the other hand, if my task is to build a bookshelf for the house, I would select different tools than those I use for cooking. I could use a hammer, a screwdriver, a saw, nails, and screws. Similarly, I can further narrow my tool selection at any point based on the step I'm at in the project. Early in the project, I might use a saw to cut the wood, but later I would use the hammer and nails to assemble the bookshelf.

To select tools for projects around my house, I determine what I'm trying to accomplish and then decide which tools best help me meet those goals. There are many RML models, and selecting the right models to use for specifying a solution is similar to selecting tools for home projects. The task can be overwhelming if all you have is a list of models without a framework to help you narrow the list.

This chapter will help you select models for your project based on the project phase you are executing and the project characteristics. For each of these factors, we overlay the model categories to ensure that you consider all types of models.

Selecting Models by Project Phases

Models are a key component of every stage of the software process. It's important not to lose sight of the overall project approach and become bogged down in the minutiae of creating models, but rather to understand how the models fit into the approach to ensure a complete set of requirements during the entire development cycle. Figure 25-1 shows a generic requirements process and the activities that occur within each phase. The phases are envision, plan, develop, launch, and measure. Within each phase there are requirements-based activities. These process phases map to phases in virtually any development approach (such as waterfall, iterative, agile, and custom approaches). The focus is on the activities within the phases rather than how the phases are executed. The key is that no matter what your development approach is, within each phase there are activities that benefit from using requirements models.

Using the appropriate models helps all stakeholders understand the requirements so that nothing important is missed and only the requirements that are important are implemented. The following sections further discuss the role of models in each phase of the requirements process. Although these are described in an order similar to a waterfall approach, they can be applied in any development methodology.

The information from the following sections is summarized in a table in Appendix A, "Quick Lookup Models Grids."

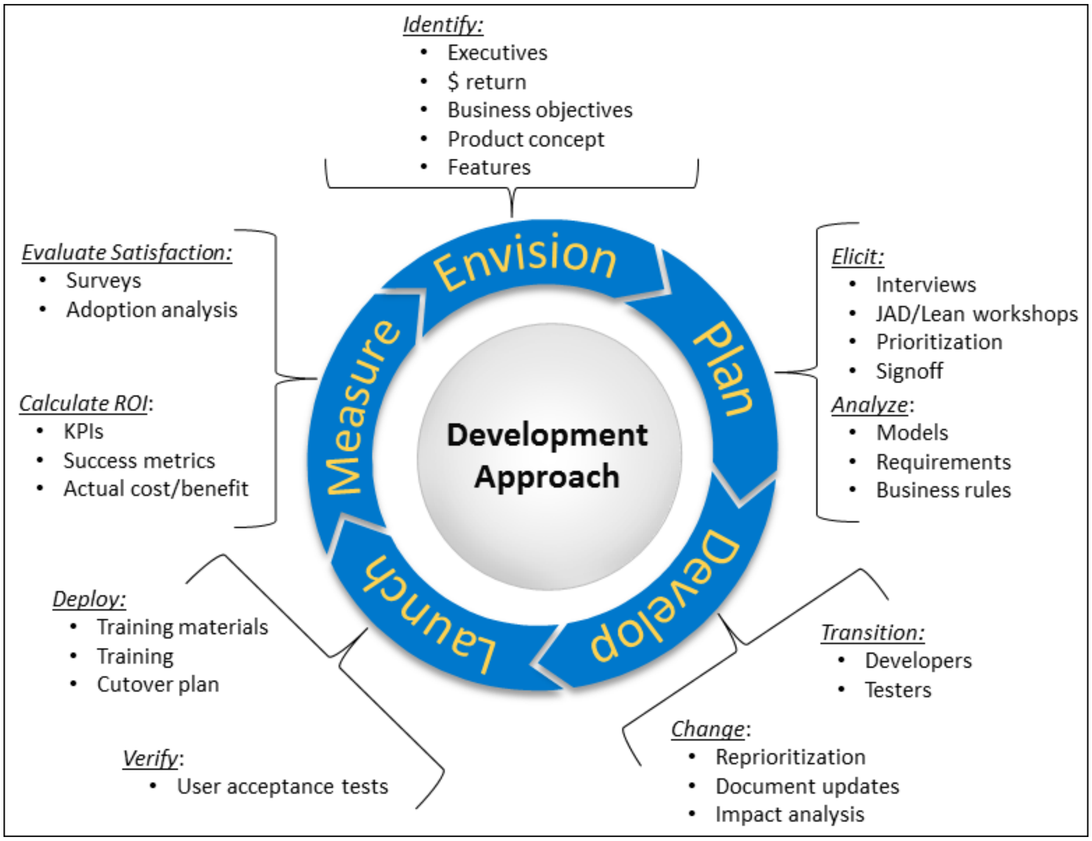


FIGURE 25-1 A requirements process.

Envision Phase

The Envision phase occurs before a project is actually chartered. Executives who own corporate strategy and manage a portfolio, program, or product roadmap have to determine which projects to fund based on the value they add to the organization. This phase determines how the project supports the corporate or program strategy, what value the business stakeholders will receive from it, and what high-level features the stakeholders need to achieve that value.

During the Envision phase, a business analyst helps executives explore the business problems, determine the business objectives that all requirements eventually map back to, develop the business case for the project, and set a broad scope for the project. The business problems and objectives are captured in a Business Objectives Model, and the highest-level features are captured in a Feature Tree. The business objectives enable business analysts to determine the project's priorities, which will be used throughout the entire project to help the team make decisions when cutting scope or when trying to decide which features to develop first.

You should also create initial high-level models such as Org Charts and Ecosystem Maps. Org Charts are a great starting point for creating people models, because they offer the opportunity to consider every single person or role that might affect or use the system. Though Ecosystem Maps do not identify specific requirements, they do identify the interfaces for which to elicit requirements and the systems that might be affected by the project. In addition to models, you should create an issues list to log every outstanding requirements-related unresolved issue or question. If you are not using a requirements management tool, then typical documents that you might produce during this phase might have names such as "business case," "business requirements document," "marketing requirements document," "product backlog," and "vision scope document."

Models in Agile

Many organizations try to depict agile methods as mini-waterfalls. Even the requirements process diagram in Figure 25-1 could be interpreted as a mini-waterfall. However, it should not be interpreted in that way. The diagram is simply intended to classify the activities that a business analyst would perform regardless of the development approach. In a waterfall approach, each phase would require signoffs before the project could enter the next phase. In an agile project, all of the phases could be executed simultaneously within a particular sprint. However, even when you are using an agile approach, at some level (perhaps at the individual story level) you have to figure out the value of the story first (Envision); then the details of the story, such as acceptance criteria (Plan); then your developers build it (Develop), potentially updating the story as you gain a better understanding of the project; and then you test and deploy what you have just built (Launch). Within a sprint these phases might be occurring every single day, simultaneously.

Even when you are using an agile approach, on very large projects with many teams that have to synchronize their work, develop some of the models up front before any development begins. This is the best way to ensure that each team has a shared understanding of the overall business goals and system requirements. Models such as the Business Objectives Model and the Objective Chain help to ensure that all teams really understand the value of the project. Org Charts, Ecosystem Maps, Business Data Diagrams, and Process Flows ensure that all teams have a shared understanding of how users will need to use the system and the environment that they are working in. A product backlog is great for managing the project, but it provides no framework for determining which things need to be in the backlog in the first place. Models can be used to populate the backlog and determine any detailed requirements and business rules within each sprint.

Plan Phase

During the Plan phase, you are trying to determine how the software needs to work so that it will achieve the value expected by the business. By listing out all the requirements, including the functions the software needs, the business rules that influence the functions, and the non-functional qualities, you will create a complete list of requirements that your business stakeholders, developers, and testers can use to build, configure, or test the system to ensure that the business value is achieved. Identify which models are needed based on the characteristics of the project, and define a requirements architecture (described in Chapter 26, "Using Models Together"). Finally, you might select tools to create and store models that support the requirements architecture, and create a requirements plan that outlines when each of the models will be created in relation to the others.

You should test out your requirements architecture and process on a portion of the project, adjusting it as you determine what works best for your organization. Project priorities can change; analysts might find that the choice of models is not sufficiently capturing requirements, or a variety of other reasons can force the team to alter the requirements architecture. In these scenarios, it's important to review the already created requirements and models to ensure that no additional artifacts need to be created or existing ones altered; changes to the requirements architecture often affect prior work.

This phase is where you will do most of the elicitation and analysis to complete models, working from a high level to more detailed models (see Chapter 26 for more about this). You also should derive requirements from models. In addition, you should continue to update your requirements issues list as you determine parts of models that you cannot yet complete.

During this phase, you might create a Key Performance Indicator Model (KPIM) to articulate how the project will improve or at least maintain the throughput of the business. You will also create Objective Chains to determine the scope of the requirements. You should create Business Data Diagrams (BDDs) and Process Flows to define and bound the project scope. The remainder of the models you need on the project will typically be created in this phase.

Further, you can use the Business Objectives Model to prioritize the analysis work you have to complete. If you are not using a requirements management tool, examples of documents that you might create during this phase include business requirements documents, system requirements specifications, software requirements specifications, functional requirements specifications, sprint backlogs, and user stories.

Develop Phase

After the models have been created, validated, and verified, the next step is to ensure that the development and testing teams understand what they need to build. Developers and testers will use models and corresponding requirements and business rules to build code, configure existing systems, and develop test cases. During your model creation, you should have derived requirements for most of the models. This is the phase for which it is most important to have done that step, because developers and testers have an easier time knowing that their job is done when they have a list of functional requirements and business rules that they can use as a "checklist" rather than just a model to build and test from.

During this phase it is important to fully explain the usage of the models to these teams; otherwise, the models might be interpreted and used in a manner for which they were not intended. For example, developers might use only the Use Cases and not look at the requirements if they do not understand how the two fit together. Further, they might look at only the requirements and not the models, missing the important context you set with the models.

It is common for this phase to also include steps for updating models that are now in a "locked" state as changes occur. This means that all changes to requirements and models must be approved and communicated to teams downstream of the business analysts because those teams might have created artifacts using those requirements and models.

During this phase, you should be maintaining your issues list as well as updating all documents and models as necessary. You should be engaged with the developers as they build the solution, and with testers as they ensure that the system is built properly. Your role is to clarify how the software should work, update priorities, and ensure that what is being built works the right way. No matter how well you document the requirements, there will always need to be verbal clarifications, especially as you get into the details of the business rules, usability, and detailed functionality of the system. As you explain the functionality, you will most often be using models to help other team members understand the context. For example, you could use a Process Flow to help the developers understand what the user is trying to accomplish, then use a BDD to explain how the business thinks about the data elements. During the development phase, you will be updating your existing documents and might create documents such as "user acceptance tests" or versions of previous documents tailored to specific development teams.

Launch Phase

During the Launch phase, the business stakeholders confirm that the solution meets their needs. They can use Process Flows and Use Cases to create user acceptance tests to perform this verification. When the system has stabilized to the point where the business evaluates and accepts the solution, it is ready to be deployed. Process Flows, Use Cases, Roles and Permissions Matrices, and Display-Action-Response (DAR) models can be used to create training materials for the new system. Typical documents created during this phase could include training manuals, user guides, help files, and any other materials that will ensure that users have high satisfaction and adoption rates.

Measure Phase

The Measure phase takes place after a new system is live and the users have adopted the system for their use. During this phase, analysts can measure the return on investment of the solution's business objectives and use key performance indicators (KPIs) to truly determine the value that the project brought to the organization. By using real data and a live measurement, it is possible to confirm that the business objectives set at the start of a project in the Business Objectives Model were actually achieved. Furthermore, the organization can measure individual business processes to ensure that they have met KPI targets described in KPIMs. Documents created during this phase might include presentations to executives describing the return, lessons learned documents, or retrospectives.

Selecting Models by Project Characteristics

In addition to the project phase, you should take into account the characteristics of the project when selecting which models to use. Typically, the first questions to ask are:

- Is this system being custom built from scratch, or will it be purchased from a vendor?
- Will the system replace an existing system entirely, result in a new implementation, or enhance an existing system?

There are several common project characteristics that can be used as guidelines to help determine which models to use. The list of project characteristics in the sections that follow is not meant to be comprehensive but is meant to give you a starting point for determining which models might be appropriate for your project. Project characteristics are not mutually exclusive, so you probably will have multiple characteristics that apply. For example, a project to replace an existing system can also be a cloud implementation. A system that will exist in a large ecosystem might also be an analytics system.

A Model to Select Models

Notice that we actually use a model to communicate how to select models by project characteristics. We use a grid that has all project characteristics down the left side and all models across the top. However, leaving the grid in that form would have given us 20 items in the list of characteristics, which is far more than 7+/-2 items. To make the grid more consumable, we further divided the characteristics into Objectives, People, Systems, and Data categories. Each project characteristic section in this chapter shows the relevant row of that grid. The complete grid is in Appendix A, for your quick reference later.

To select models using this section, first decide which project characteristics apply to your project, and then consider the suggested models for those characteristics (shown in rows in the grid in each section) to determine which models will be helpful on your project. The recommended models grid shown in each project characteristic uses the key in Table 25-1.

TABLE 25-1 Key for Models by Project Characteristic

Meaning	Cell Value
Likely to be needed	L
Might be needed	M
Not needed based on this characteristic alone	blank

The meaning of blank cells is tricky and is important to understand so that you can interpret this information correctly. If a cell is blank, it means that the particular characteristic alone does not indicate the need for that particular model. The project might still need the model, though, because of another relevant project characteristic. Conversely, if the cell is filled in, it means the characteristic alone is sufficient to indicate that the model is needed for that type of project. For example, you might notice that the Business Objectives Model is not indicated for most of the characteristics. That's because the use of a Business Objectives Model is dependent on a project having one of only a few of the characteristics; however, those characteristics are actually pretty common, and at least one of them will apply to almost all projects. Also, a project for implementing enhancements to replace existing systems functionality might need a KPIM, but it needs the KPIM because the project is replacing an existing system, not because it is an enhancement.

Objectives Characteristics

The following objectives characteristics help determine which objectives models are needed based on the type of project implementation.

Greenfield Projects

A greenfield project is one in which a brand new system is custom built from scratch because no system currently exists to provide the needed functionality. A primary consideration in these projects is scope, because it can easily balloon out of control. Many future users of these systems will offer input on features and requirements that they have for the system, and it is important to prioritize these requests in the context of a Business Objectives Model and Objective Chains. You should create a Feature Tree to show all planned features, and then tie each feature to a business objective so that the organization only builds the features that have the most value. A Requirements Mapping Matrix (RMM) is important to ensure that requirements ultimately map back to business objectives through other models.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Greenfield		L	L		L	L																				

Commercial Off the Shelf (COTS) Projects

The goal of a COTS project is to evaluate and select a third-party solution to solve a business problem and then implement it. We have broken the COTS project characteristic into two aspects, selection and implementation, because in many organizations they exist as completely separate efforts with differing model needs. Often the selection phase proceeds, but then COTS implementation does not occur because the team decides to build a new system or improve the existing system.

COTS selection The selection phase involves qualifying vendors, determining what the primary business objectives are, and ultimately selecting a system that meets an organization's needs. Creating a Business Objectives Model and Objective Chain might be important to ensure that the selection process focuses on the systems that provide the greatest return on investment (ROI) as defined by the business objectives.

During the selection phase of a COTS system, many organizations create a list of features that the organization needs. This method is extremely weak because it does not address how the software satisfies the business process. To resolve this, you should use Process Flows and KPIMs to prioritize the Process Flows and determine the features that support the most critical business processes. You should use an Org Chart to ensure that you are talking to all of the right people about their Process Flows. During interviews with the vendor, you can use the highest-priority Process Flows to have the vendor demonstrate exactly how those processes and KPIs would be satisfied by using their software. You might still want to create a Feature Tree to help you quickly summarize the features deemed to be most important. An RMM helps you keep your features prioritized. In addition, a BDD should be used to ensure that there are no major discrepancies in data models between the software and the business needs. A Data Dictionary might not be needed because the individual fields typically do not require major changes. Finally, an Ecosystem Map should be used to ensure that there is a good understanding of the integrations that the COTS system will need to support. The vendor should be prepared to address gaps between the models and the COTS system.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
COTS selection		M	M	L	L	L		L	L				L								L		M			

COTS implementation The implementation phase of a COTS project could be part of an existing system replacement project, but it could also represent installation of a completely new system. If the COTS system is replacing an existing system, and there is little customization, KPIMs are better because they help ensure that business throughput is maintained at desired levels. If there is significant customization, then there are features that you can map to business objectives by using a Business Objectives Model and Objective Chain. If the COTS system is not replacing an existing system or is introducing a significant amount of new functionality, defining the business objectives and their relationship to features by using a Business Objectives Model and Objective Chain is crucial to prioritizing features.

Org Charts can help ensure that all existing users are represented. Process Flows ensure that those users' functionality needs are understood. The RMM is helpful for ensuring that all requirements of the business processes are implemented in the new system. A Roles and Permissions Matrix is useful because many COTS systems allow you to configure roles out of the box, so this helps you decide who should have those roles and what permissions those roles should have. Ecosystem Maps and System Flows are important if the COTS system will be deployed in an existing ecosystem, to help identify integration points. BDDs and Data Dictionaries help ensure that data used in existing systems or processes is considered, whether it is to be converted to a new type of data or not. Most COTS software comes with standard reports, so Report Tables are important to define how those are deployed. Use Cases might be used to describe how users will interact with the COTS software directly. DAR models might be useful because tables can be created for each element in the UI that is configurable, to help capture the configuration requirements.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
COTS implementation		M	M	L		L		L	L	M	L		L	L		M					L		L			L

Enhancement Projects

An enhancement project makes a major change to an existing system by adding new functionality to the system's capabilities. Because enhancement projects tend to primarily focus on new capabilities, focusing on mapping features to business objectives is paramount to minimize gold plating. The Business Objectives Model and Objective Chain are extremely important, because they allow you specifically to restrict the solution's scope to the appropriate people, systems, and data. The RMM will help you continue the traceability by mapping requirements to other prioritized models to control scope. The Feature Tree can provide a quick view of all features that are in scope for implementation.

Because an existing system is being modified, often the data model is not changing. In these cases, a BDD is not necessary. By the same token, if the new features do not require additional integration, then an Ecosystem Map might not be necessary either.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Enhancement		L	L		L	L																				

People Characteristics

The following project characteristics are related to the users who use the system.

Systems with Extensive User Interaction

Systems with extensive user interaction are defined as those that have many users performing many types of operations in the system. You should focus on people models first. Typically, Process Flows will be used in a project that is heavily driven by the user interface (UI). Use Cases might be helpful to further describe user interactions. Roles and Permissions Matrices are helpful if you have a limited number of roles or types of user and need to implement a security model in the user interface. UI Flows and Display-Action-Response (DAR) models will be two of the most important models, because they illustrate visual aspects of the solution that Process Flows and Use Cases cannot capture. Each step in the Process Flows can then be linked to a DAR model to illustrate the UI in the level of detail needed.

If you are replacing an existing system or automating a process, KPIMs will probably be helpful because of the increased focus on end-user throughput and completion of tasks. If your project is an enhancement with an extensive UI, these models will be key in order to allow you to keep a consistent look and feel with the current software.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Extensive user interaction									L	M	L				L	L										

Customer-Facing Systems

Customer-facing systems primarily have users that are external to the organization that is implementing the system. There are typically a few internal roles as well, such as administrative roles, but most users are from outside the organization.

Because the users are external, Org Charts are typically not helpful. Roles and Permissions Matrices might be used to define the type of permissions necessary for security in the system. Process Flows and Use Cases might be important to describe how the customers will use the system. UI Flow and DAR models should be created to ensure that the user interface is easily navigated and used by external users. This is important even if there aren't very many customer-facing screens.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Customer-facing									M	M	M				L	L										

Business Process Automation Projects

Business process automation projects either fully or partially implement the business's processes in a system. These projects use KPIMs because the performance of existing manual processes can be measured to ensure that the performance levels are either maintained or improved in the new system. Org Charts are important to ensure that you talk to all existing business stakeholders who execute the process. Process Flows can be used to document existing business processes that will be performed using the new system. Use Cases might help describe how those activities will occur in the system. Furthermore, future-state Process Flows and Use Cases can illustrate how the system should function and provide a basis for new training guides.

Roles and Permissions Matrices might be important for putting a security model in place. BDDs and Data Dictionaries will be necessary to describe the business data objects and fields used and manipulated in the process.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Business process automation				L				L	L	M	M										L		L			

Workflow Automation Projects

A workflow is a specific type of business process that has a heavy emphasis on approvals and routing of information between groups. A project that automates a workflow typically requires a Process Flow to describe context for the workflow. These projects usually have a BDD to show the business data objects that are manipulated during the workflow. State Tables and State Diagrams can help show how the objects change state during the workflow. Typically, these projects have security needs related to who can perform functions at different steps in the workflow, so Roles and Permissions Matrices are helpful.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Workflow automation									L		L										L			L	L	

Systems Characteristics

The following project characteristics are related to the type of system being worked on.

System Replacement Projects

A system replacement project replaces an obsolete solution with either a custom-built system or a COTS system. When an existing system is being replaced, the business objectives are often related to goals such as improving throughput, performance, reduction in license fees, or reduction in maintenance costs. These goals are often not achieved through implementation of specific new features. Even if there are new features, a majority of the functionality simply needs to be maintained as compared to the legacy system. The Business Objectives Model and Objective Chains aren't as useful because their purpose is to map the value (for instance, return on investment) of features to business objectives. Unlike projects that trace their value up to business objectives, existing system conversion projects should use KPIMs for business processes to prioritize requirements and business rules. At a minimum, the new solution must be able to maintain the KPIs at their current levels—there should be no degradation in overall efficiency from switching to a new solution.

The KPIM is one of the most critical models because it helps analysts demonstrate to business stakeholders that even if the new system behaves differently, the business outcomes will be the same or better. One common challenge with existing system conversions is that new software might cause a reduction in the KPIs of one group while improving the KPIs of another group. Even though overall the throughput and business value are positive, the group that is negatively affected might not approve the system unless they understand that the negative impact is in the context of an overall improvement to the business. You should use KPIMs to reassure the business stakeholders that the new system, though different, will still let them get their jobs done.

With the use of KPIMs, you will also need Process Flows against which to map the KPIMs. Org Charts, Ecosystem Maps, and BDDs are all valuable to an existing system conversion project as well, because they help you understand all current users, the existing system integrations that might need to be replaced, and the full set of data the business cares about. Process Flows are needed to describe the activities that users perform in the existing system. Report Tables are needed because existing systems almost always have reports that need to be converted to the new system.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
System replacement				L	L	L		L	L	M	M		L	M	M				M		L	M	L			L

Real-Time and Embedded Systems

Real-time and embedded systems have significantly smaller or more primitive user interfaces than most user-facing systems. The goal of these types of projects might be to implement automation or controller systems. In real-time and embedded systems, System Flows are the dominant models. Ecosystem Maps and System Interface Tables might be helpful if the real-time system has interfaces with many other systems. Most people models will not be helpful, because a majority of the effort will focus on the steps within the system. Real-time and embedded systems often have very simple data models, so a BDD and a Data Dictionary might not be necessary. Although the system will obviously be dealing with data, it is most likely doing so at a technical level, so the business stakeholders are not concerned about the details of the data itself. State Tables and State Diagrams are commonly used, because these types of systems often have complex state changes that trigger behaviors.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Real-time and embedded system													M	L					M			M		M	M	

Large-Ecosystem Projects

Projects with large ecosystems have many existing systems that interact. You should focus on the systems first by starting with an Ecosystem Map. System Interface Tables might be needed to describe the interface requirements between systems. Identify the business data objects to create BDDs, and then use Data Flow Diagrams (DFDs) to show the flow of data between the systems. Data Dictionaries will be necessary to describe the fields and rules for the data.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Large ecosystem													L						M		L	L	L			

Internal IT Systems

Internal IT systems are systems in which all (or most) of the users are internal to the organization. These systems are deployed in one organization's environment. Org Charts will certainly be used because the users are internal, and Process Flows are necessary because they define how the business will use the internal system. Roles and Permissions Matrices will probably be used to describe the security model for users. Ecosystem Maps are helpful to show how the system fits in with other existing systems in the IT organization, and System Interface Tables might be helpful if the interface requirements are important to the business stakeholders.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Internal IT								L	L		L		L						M							

Hardware and Software

Systems that have both hardware and software components to be implemented typically have many inputs and outputs to be considered. Although people and data models are important, system models are the most critical to create for this characteristic. An Ecosystem Map shows how the components are related, System Flows show how the hardware and software interact, and System Interface Tables describe the inputs and outputs between each component. Keep in mind that although these models are similar to technical models, the purpose of these models is to derive the requirements—whatever the business stakeholders need. Leave the technical documentation to the technical team.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Hardware and software													L	L					L							

Packaged Software

Packaged software is software that is sold as stand-alone software. Packaged software should heavily use people and data models and will probably use few system models. Process Flows and Use Cases are helpful for showing how the users will interact with the software. Org Charts are not useful because the users are not common to one environment. Feature Trees are useful for actually building the "packaging" for the software. RMMs help control scope by mapping the requirements to Process Flow steps to ensure that extra features that are not anticipated to create significant value for the users are excluded. UI Flows and DAR models are important for modeling the user interface.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Packaged software					L	L			L	M					L	L										

Cloud Implementation Projects

In cloud implementation projects, you are implementing a cloud solution to solve a business problem. The fact that the project is a cloud implementation does not influence the selection of requirements models as much as other the other characteristics do. For instance, if the cloud implementation is part of a large ecosystem, then an Ecosystem Map will be useful to show how the cloud part of the solution interacts with other parts of the system, a System Flow can help describe the system steps, and System Interface Tables might be necessary to describe the interfaces. The Org Chart might be useful in identifying users, and Roles and Permissions Matrices can define security access in the cloud for the user types. Process Flows can be used to describe how users will interact with the cloud. Also, state models are often useful because cloud implementations typically are based on user states, such as logged on, logged off, online, or offline.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Cloud implementation								M	L		L		L	M					M					L	L	

Web App Projects

Web apps expose functionality and display data to users through a web interface. Because the web app is communicating to a back-end server, an Ecosystem Map is helpful for showing the architecture, but System Flows are most important for describing those interactions. Data models are important for showing what data exists in the system and is passed between the server and client, so you should create a BDD and a Data Dictionary. UI Flows and DAR models are often helpful for building the web interface, to ensure that it is usable.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	People	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	Systems	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	Data	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Web app													L	L	M	M					L		L			

Mobile Systems

Systems with mobile capabilities are intended to be deployed at least partially on mobile devices. Mobile systems should have Use Cases that precisely describe how users will interact with the mobile device, and they might use Process Flows to describe users' goals with the device. Ecosystem Maps and System Flows might be helpful to show the interfaces and interactions between the mobile system and the servers it communicates with. Because mobile devices have limited screen size and sometimes slow interaction times, UI Flows and DAR models are helpful to ensure that the screens on the mobile device are designed efficiently and can be easily used.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	People	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	Systems	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	Data	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Mobile									M	L			M	M	L	L										

Projects with Complex Decision Logic

Projects with complex decision logic automate a decision process. These projects typically have other characteristics that drive model selection. You should use Decision Tables and Decision Trees to model the complex logic. State Tables and State Diagrams might be needed because many decisions are based on states in the system.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	People	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	Systems	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	Data	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Complex decision logic																	L	L						M	M	

Data Characteristics

The following project characteristics are related to the data needs of the system.

Analytics and Reporting Components

Systems that have analytics and reporting components are typically used in business intelligence to help people make decisions based on large data sets. In fact, these projects can be identified by their business strategy—any project that involves getting information to make a decision has significant data requirements.

Projects that involve significant volumes and use of data need several data models to accurately document the requirements. You can use the BDD to determine which types of data are involved in the project, DFDs to describe the flow of the data, and Data Dictionaries to further describe the data. Report Tables are a necessity if you are creating reporting requirements.

Often Process Flows and the other people models aren't needed at all for a pure analytics project. However, keep in mind that Report Tables include decisions that need to be made. For a very large business intelligence project, prioritizing reports might require Process Flows for determining which reports support the most important processes and for articulating the decisions that need to be made. Also, decision models might be helpful on analytics projects.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Analytics and reporting									M								M	M			L	L	L			L

Database Back-End Components

Many projects will have a database back-end component. These systems contain data that is used by and stored in the system. For these projects, you will need to identify all of the business data objects in BDDs and define their flow between processes, systems, and storage components in DFDs. You should also define the actual field-level details in Data Dictionaries. Keep in mind that you do not need to document the database schema or physical architecture of the database servers. Instead, focus on documenting how the business stakeholders think about the data.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Back-end database																					L	L	L			

Data Warehouse Components

Data-heavy systems contain many business data objects and pass a large volume of data between systems. You will probably use many non-data models, but you should focus on data first, by identifying business data objects to create BDDs, then completing DFDs. You can create Data Dictionaries to provide an increased level of detail for the data requirements.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Data warehouse																					L	L	L			

Project Examples

An existing financial services system is to be replaced with a heavily customized COTS product that will integrate to several other existing systems. The system will be used by hundreds of thousands of customers every day. The team works with several departments that handle the various regions that are transitioning to the new system. A majority of the functionality will be maintained. Figure 25-2 shows the appropriate characteristics and most useful models for this project.

Another project is to build a single-user game for a mobile device. Figure 25-3 shows the appropriate characteristics and most useful models for this project.

A final project is to select and implement COTS software to manage a loan approval process. Figure 25-4 shows the appropriate characteristics and most useful models for this project.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
COTS implementation		M	M	L		L		L	L	M	L		L	L		M					L		L			L
Extensive user interaction									L	M	L				L	L										
Customer-facing									M	M	M				L	L										
System replacement				L	L	L		L	L	M	M		L	M	M				M		L	M	L			L
Large ecosystem													L						M		L	L	L			
Database back-end components																					L	L	L			
Selected models for scenario		x	x	x	x	x		x	x		x		x	x	x	x					x	x	x			x

FIGURE 25-2 Models for a financial services example project.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
Customer-facing									M	M	M				L	L										
Mobile									M	L			M	M	L	L										
Complex decision logic																	L	L						M	M	
Selected models for scenario									x	x					x	x	x	x						x		

FIGURE 25-3 Models for an example mobile game project.

	Objectives	Business Objectives Model	Objective Chain	Key Performance Indicator Model	Feature Tree	Requirements Mapping Matrix	**People**	Org Chart	Process Flow	Use Case	Roles and Permissions Matrix	**Systems**	Ecosystem Map	System Flow	UI Flow	Display-Action-Response Model	Decision Table	Decision Tree	System Interface Table	**Data**	Business Data Diagram	Data Flow Diagram	Data Dictionary	State Table	State Diagram	Report Table
COTS selection		M	M	L	L	L		L	L				L								L		M			
COTS implementation		M	M	L		L		L	L	M	L		L	L		M					L		L			L
Workflow automation									L		L										L			L	L	
Complex decision logic																	L	L						M	M	
Selected models for scenario		x	x	x	x	x		x	x		x		x	x			x	x			x		x	x	x	x

FIGURE 25-4 Models for an example loan approval project.

Thinking About the Audience

Consider who the audience is when you select your models. All RML models are designed to be understood and used by all audiences. However, when you ask someone to review or use models, you should still select models that are most appropriate to them.

- Asking a vice president to review models that are very detailed might not be a good use of her time. Conversely, creating only very high-level models for developers and testers to use will not provide them with enough information to do their jobs.
- You might find that business stakeholders have a hard time telling you all of the systems and their integrations, whereas architects might not be very familiar with business processes or how the product manager intends users to use the system.

Regardless of who creates, reviews, and uses the models, you should make sure that your development teams are aware of the full set of models and requirements that you do create. Models linked to requirements provide additional information beyond just the checklist of requirements to be developed. Although models represent a way to organize and present information, it is still necessary to verbally communicate with all stakeholders to ensure that they understand the material. Handing over models without discussing them with the technical teams is a recipe for failure. You will never be able to capture every iota of information in the models.

Table 25-2 describes the most common stakeholder audience scenarios. The types of stakeholders who will directly help with creating a model are marked with a C, those who will be more likely to only review what you give them are marked with R, and those who probably won't use it at all are blank. The analyst is not listed in this table because it is assumed that he will help create and review all models.

TABLE 25-2 Model Use by Audience Type

Model	Business	Technical	Executive
Objectives Models			
Business Objectives Model	C	R	C
Objective Chain	C	R	R
Key Performance Indicator Model (KPIM)	C	R	R
Feature Tree	C	R	R
Requirements Mapping Matrix (RMM)	R	R	
People Models			
Org Chart	C	C	R
Process Flow	C	C	R
Use Case	C	R	
Roles and Permissions Matrix	C	R	

Model	Business	Technical	Executive
Systems Models			
Ecosystem Map	R	C	R
System Flow	R	C	
UI Flow	C	C	
Display-Action-Response (DAR) model	C	R	
Decision Table	C	R	
Decision Tree	C	R	
System Interface Table	C	C	
Data Models			
Business Data Diagram (BDD)	C	R	
Data Flow Diagram (DFD)	R	C	
Data Dictionary	C	C	
State Table	C	C	
State Diagram	C	C	
Report Table	C	C	R

Tailoring Models

When you are selecting requirements models, it is possible to get stuck trying to fit information into a structure that is not appropriate. It might be necessary to make small adaptations to the structure of some models based on the specific needs of a project. Each requirement model is flexible enough that the components can be tailored to the specific needs of the project.

The most common types of customization we see are the addition of coloring to highlight particular types of elements or the addition of elements, such as fields to a Data Dictionary or a particular shape from Business Process Modeling Notation (BPMN) to a Process Flow.

Avoid modifying the models unless it's absolutely necessary to communicate additional information that the model does not normally contain. If you do need to modify them, do it in such a way that an untrained user can still consume them easily. If a model must be modified, the tailoring should take place before the start of the project rather than during the requirements process, to reduce the amount of rework and to avoid inconsistency in the requirements documentation. Often you might not recognize that a model needs to be modified until you have done a significant amount of work. In those cases, you will have to use your best judgment as to whether it is worth it to go back and fix the prior work.

Exercise

The following exercise is intended to help you to gain a better understanding of how to use the information in this chapter. The exercise is open ended, and therefore the answer you come up with could be substantially different than the answer that we have provided. There are potentially many correct solutions. The answer provides an explanation of how we arrived at our solution. You will gain the most out of the exercise by attempting to do it yourself before looking at the solution. The answers for the exercises can be found in Appendix C.

Instructions

Identify the project characteristics and select the appropriate models for the scenario.

Scenario

You are on a project to launch a brand new eStore to sell flamingos and assorted lawn decorations, and you have to document all of the requirements. Currently, orders are only taken over the phone and manually entered into the order system by sales representatives. You expect to have thousands of customers visiting the website every day. You know that there will be servers that push catalog data to the website and send the orders on to fulfillment.

As you explore the high-level features, you also learn that the eStore orders will have various states, such as “New,” “Received,” “Packaged,” “Billed,” “Shipped,” and “Returned.” The president of Wide World Importers wants to view reports that show him metrics such as sales volume, inventory volume, and inventory costs by month. In addition, the training team wants to be sure that the process of shopping is well documented from the point at which a customer arrives at the website to when he receives an order confirmation after checkout.

Index

About the Authors

JOY BEATTY is a Vice President at Seilevel. Joy drives creation and implementation of new methodologies and best practices that improve requirements elicitation and modeling. She assists Fortune 500 companies as they build business analysis centers of excellence. Joy has provided training to thousands of business analysts.

Joy is actively involved as a leader in the requirements community, serving on boards of multiple industry organizations. She is currently on the International Institute of Business Analysis (IIBA) core team for updating *A Guide to the Business Analysis Body of Knowledge (BABOK Guide)*. She has presented at numerous requirements-related conferences and speaking events. Additionally, she writes about requirements methodologies in journals, white papers, and blog posts. Joy graduated from Purdue University with Bachelor of Science degrees in both Computer Science and Mathematics.

ANTHONY CHEN is President of Seilevel. Over the past 15 years, Anthony has worked with numerous Fortune 500 companies. He is responsible for the strategic growth of Seilevel and the development of innovative software requirements techniques, including Objective Chains; the Business Objectives Model; the Objectives, People, Systems, and Data classification (OPSD); and RML.

In addition to his business and innovation leadership, Anthony has written extensively on software requirements techniques, experiences, and ideas. Some of his writing can be found on the Seilevel blog. He earned Bachelor of Science degrees in both Electrical Engineering and Microbiology from The University of Illinois, and a Master of Science in Medical Microbiology and Immunology from Texas A&M University.

You can contact us by leaving a comment on the Seilevel blog (*http://www.seilevel.com/blog/*), emailing us at RML@seilevel.com, or joining our conversation on Twitter @seilevel.