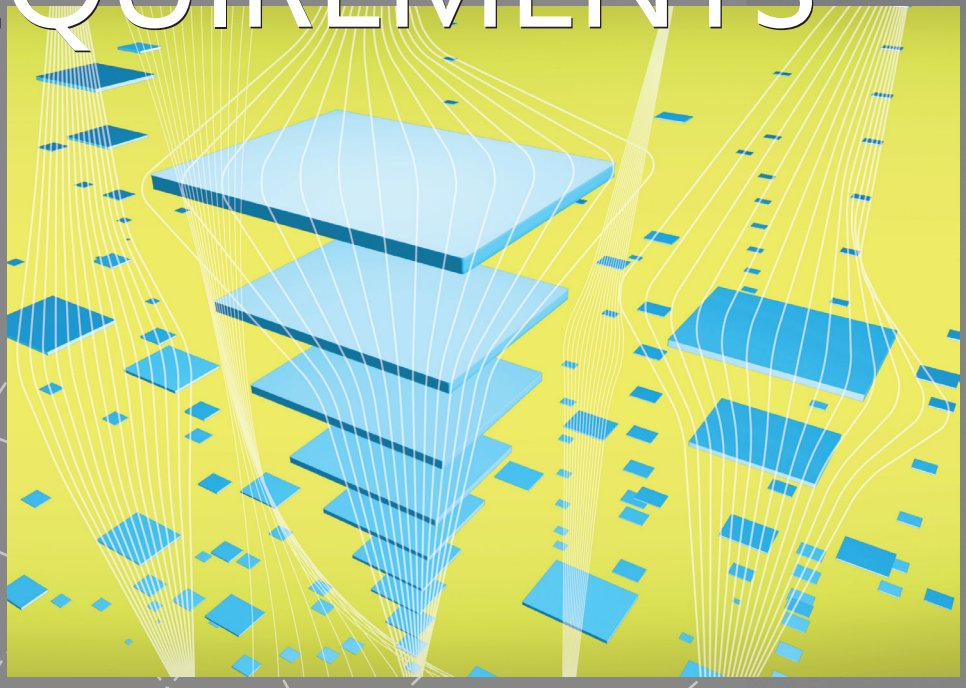
Microsoft

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.

Second Printing: March 2015

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at http://www.microsoft.com/learning/booksurvey.

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.

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*.

PART I

An Introduction to Models

CHAPTER 1

Introduction to RML

With nine months until the holiday season, a prominent online retailer had identified a critical set of new features to add to its website that would greatly enhance the customer experience, directly increase sales, and reduce customer support calls in multiple countries simultaneously. The features were estimated to be worth $14 million a year and to cost less than $2 million. The product manager identified the requirements and business rules for the features, and the development team and project management team created estimates that showed that the project team would easily be able to finish by the start of the holiday season. The team was driving hard to the end date, often working nights and weekends to get releases out the door.

After eight months, the team was in final testing and feeling pretty good. They had completed a very long list of enhancements required to achieve the very hefty return. Then one of the testers noticed that the tax calculations were not right. Unfortunately, those calculations were just the tip of the iceberg. It turned out that the team had neglected to talk to the tax team. In fact, they hadn't even realized that it would be necessary to do so. If they had, they would have discovered that the tax rules required to operate in several of the countries were extremely complex, requiring integrations to third-party software that managed the rules. The project was delayed, and the $14 million return was lost for that season. The project manager was fired, and the product manager was reassigned to a different, less prominent project.

Software projects are often plagued by missed, incomplete, or unclear requirements (The Standish Group 2009). As a result of faulty requirements practices, most projects are doomed to fail (Ellis 2008). Poor requirements are at the root of many project failures, so it is disappointing to see that the success of the software requirements industry has not dramatically improved over the last 20 years. Although academia has been steadily identifying approaches for improving requirements techniques and engineering methods, commercial practices have remained largely the same. Software programming practices have matured considerably with the creation of a variety of new techniques and a plethora of tools, but software requirements are often still developed by using long lists of "shall" statements maintained in spreadsheets. Projects that use an agile approach are rarely better, often still maintaining their product backlogs and user stories in long lists in spreadsheets or tools.

RML Defined

RML (Requirements Modeling Language) is a language designed specifically to visually model requirements for easy consumption by executive, business, and technical stakeholders. RML is not a theoretical modeling language. In developing RML, we modified existing models for ease of use and created new models to address gaps. The result is a full set of models that, at its core, is specifically designed to model software requirements and is easily adopted by business stakeholders who are often challenged by complex models. We have successfully used the requirements models on many large-scale software development projects.

Challenges with Traditional Software Requirements Practices

Traditional practices unfortunately support the use of thousands of "system shall" requirements similar to the unwieldy list shown in Figure 1-1. These requirements are typically generated via interviews and working sessions with business stakeholders. Because the average person is limited by Miller's Magic Number (see the next section, "Limitations of the Human Brain"), it is nearly impossible to read the thousands of requirements and emerge confident that the requirements are complete. Moreover, one of the more significant problems is scope creep. When you have thousands of requirements, it is difficult to determine which requirements should be cut without some way to link those requirements to values that can be compared across the solution. Teams will often organize the requirements into logical groupings, but those groupings are still usually too large to be processed effectively.

Agile approaches such as Scrum have product backlogs, user stories, and acceptance criteria. Many Scrum evangelists say that the product backlog should be an unnested list of stories, which is no better than a long list of requirements. The acceptance criteria are also supposed to be listed out, sometimes on one side of a notecard. Those who work with large systems know that this lack of information organization simply isn't feasible on a project with potentially hundreds of stakeholders.

Limitations of the Human Brain

Analysts who use traditional practices to create software requirements experience common problems during analysis, organization, and consumption of the requirements. Traditional practices use long lists of text requirements in the form of shall statements, use cases, or, more recently, user stories and product backlogs. The challenge of working with long lists of items results from a fundamental limitation of human cognition. In the 1950s, cognitive psychologist George A. Miller found that humans can only remember and process seven plus or minus two items simultaneously (Miller 1956). This is often referred to as Miller's Magic Number.

$$7 \pm 2$$

More recent evidence has suggested that the number is possibly even as few as three or four (Cowen 2001). This number represents the capacity of the brain's "scratchpad" used to hold the information to solve problems. Regardless of the actual number, if a typical person is asked to think about 15 things at the same time, up to 9 (and probably fewer) of the 15 might actually be retained and processed. If more items are being processed, they are being processed a few at a time and are quickly switched in and out of memory. Think about going to a grocery store to purchase 15 items. If you don't have a written list, it's likely that you'll return home with items missing, or with incorrect items. In the same vein, if you have a list of requirements or a product backlog that is hundreds or thousands of items long, there is simply no way your brain can make sense of the complexity unless it is broken into smaller organizational groups.

Requirements Document

REQ001	System shall have fields for firstname, middle initial and last name.
REQ002	System shall display a name if there is one in the stored profile.
REQ003	System shall require name is completed.
REQ004	System shall have a field for position or title.
REQ005	System shall require title is completed.
REQ006	System shall display a position or title if there is one in the stored profile.
REQ007	System shall have a field for email address.
REQ008	System shall have a field for alternate email address.
REQ009	System shall display an email address if there is one in the stored profile.
REQ010	System shall display an alternate email address if there is one in the stored profile.
REQ011	System shall require email address is completed.
REQ012	System shall require alternated email address is completed.
REQ013	System shall have a field for daytime phone number.
REQ014	System shall display a phone number if there is one in the stored profile.
REQ015	System shall require phone number is completed.
REQ016	System shall validate all characters in the phone number field are digits when user exits the field.
REQ017	System shall display an error message if not all characters in the phone number field were digits.
REQ018	System shall have a field for a fax number.
REQ019	System shall require fax is completed.
REQ020	System shall display a fax number if there is one in the stored profile.
REQ021	System shall validate all characters in the fax number field are digits when user exits the field.
REQ022	System shall display an error message if not all characters in the fax number field were digits.
REQ023	System shall have two fields for a street address.
REQ024	System shall require the first street address field is completed.
REQ025	System shall display an address if there is one in the stored profile.
REQ026	System shall have a field for city.
REQ027	System shall require the city field is completed.
REQ028	System shall display a city if there is one in the stored profile.
REQ029	System shall have a field for state.
REQ030	System shall display a state if there is one in the stored profile.
REQ031	System shall require the state field is completed.
REQ032	System shall have a field for zip code.
REQ033	System shall display a zip code if there is one in the stored profile.
REQ034	System shall require the zip code field is completed.

FIGURE 1-1 An example of a long list of requirements.

Pictures Are Easy, Words Are Hard

What is the solution to this fundamental limitation of our primitive mammalian brains? The adage that a picture is worth a thousand words seems appropriate. *Models* are visual representations (pictures) of information related to the processes, data, and interactions within and surrounding the solution being developed. You probably use visual models every day without realizing it.

During a recent trip to a conference in a casino, after I checked in and got my room key, the woman at the desk told me how to get to my room. She said something along the lines of "From here you are going to go out to the right, then follow the path to the left, pass the bar, pass the slots, at the fountain go right, you'll go past a restaurant, and another, then you will reach a hall where you can turn left by some shops, and at the end of that you'll find elevators near the pool entrance."

I stared at her blankly. All I could think of at that moment was the sea of slot machines and tables I had walked through just to get to the registration desk from the taxi. I assumed that I would pass many more of them on my way to my room, which would further add to the confusion of what she had just said. Then she gave me hope: "And here is a map that shows you how to get there." She had drawn the path I needed to take from registration to my elevators, much like the map in Figure 1-2. I felt relieved, because I had absolutely no capability to retain all of her directions beyond the first few, but now I at least had a model to refer back to when I got confused. A map! Simply put, when human beings interpret information, pictures are easy, words are hard.

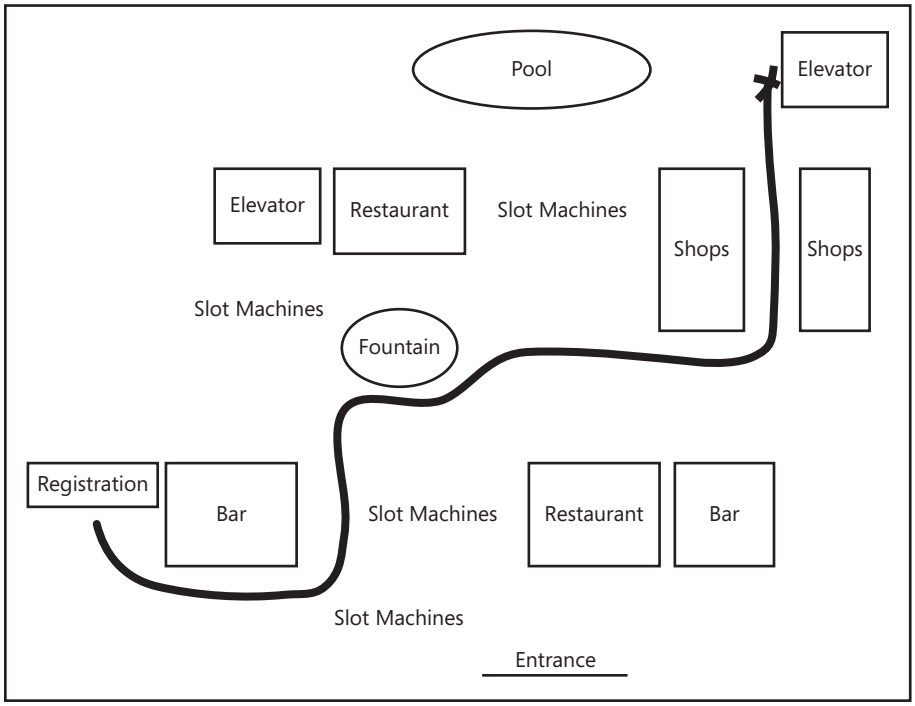


FIGURE 1-2 A map to accompany verbal directions through a casino.

Requirements Models

Requirements models organize and present large quantities of information, help you identify missing information, and give context to details (Gottesdiener 2002). Most importantly, models provide visual groupings that enable you to quickly analyze large amounts of disparate information by using limited short-term memory. It is difficult to read, interpret, and identify gaps in a requirements document of thousands of "system shall" statements, but models can help.

Imagine that you have a jumble of letters as in Figure 1-3 in front of you, and you have to find out which letters of the alphabet are missing.

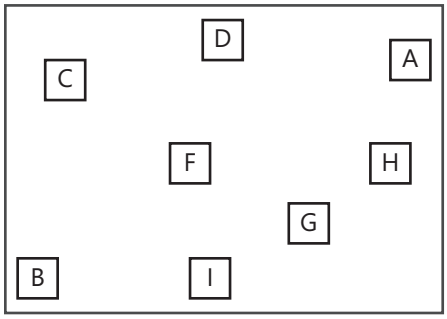


FIGURE 1-3 Jumbled letters—which one is missing?

If you just stare at the jumble or even line the letters up in a row with no order, it is going to be difficult to find the missing letters (in fact, you probably just tried to order them sequentially). If you order the letters alphabetically, as in Figure 1-4, all of a sudden the missing letter jumps right out.

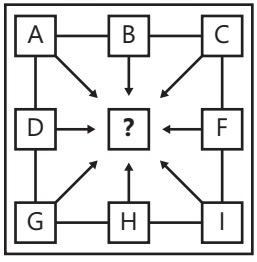


FIGURE 1-4 Organizing letters to identify the missing one.

The key to finding missing requirements is to take advantage of the fact that each requirement is related to other requirements in some way. It is extraordinarily difficult to ensure completeness when you are given a long list of "system shall" statements, but adding structure to the requirements takes advantage of their relationship and greatly simplifies the task by allowing you to analyze smaller groups of information at one time.

Requirements models are used throughout a project's life cycle. They are helpful for analyzing requirements, eliciting requirements in sessions with stakeholders, validating requirements with stakeholders, and communicating requirements to developers and testers.

Why Not UML?

The obvious question arises: Why not use the Unified Modeling Language (UML)? UML is a language used to visually specify the design of software systems (Object Management Group 2007). UML is a reasonable foundation for requirements modeling, but it is incomplete for modeling requirements because it lacks models that tie requirements to business value and models that present the system from an end user's point of view. In addition, its technical roots make it simply too complex for business stakeholders to adopt because its models are geared towards modeling the structure of the software architecture. Finally, UML is intended to be used to describe the technical design and architecture of a system and is at best retrofitted to model requirements, which are focused on business benefit, user actions, and business rules.

Models are most useful when they focus on only one or two aspects of a solution. If there are too many types of information in a model or if the syntax is too complex and difficult to understand, business stakeholders simply will not use it. In fact, our experience has shown that model complexity is one of the main reasons that some existing modeling languages are not adopted in large organizations.

RML models are designed with the simplest syntax possible that still allows the model to convey the information needed. The intent of RML is to provide a consistent syntax and semantic structures to be used by business stakeholders to analyze and understand the models on projects. The language is designed to be easy to learn and use for the entire team, including but not limited to business stakeholders, developers, and testers. The models are simplified to the most basic symbols and formats necessary to achieve the intended results within the requirements space. RML is not specific to a software development approach and can be easily adapted to work with any development approach or tool set.

Requirements vs. Design

Many of the RML models step into the realm that analysts traditionally consider to be design. For example, the Display-Action-Response model uses wireframes or screen mockups to document how a user will interact with screen elements, and the User Interface Flow shows how a user will navigate through various user interfaces.

There is a common saying related to requirements: "Requirements are what needs to be built, and design is how it will work." This distinction between requirements and design is important, because many people argue that anything that is design should not be grouped with requirements and should not be documented by business analysts. Unfortunately, there is an issue with this strict definition: "One level's requirement is another level's design."

One Level's Requirement Is Another Level's Design

At any conceptual level of a top-down solution, if you consider one level to be the "what," the next level down will be the "how." Therefore, using the what versus how definition, one level is a requirement and the next level down is design.

For example, a stakeholder might have a requirement to reduce the shopping cart abandonment rate of the company's site. At the next level of detail, a product manager might propose a few different solutions for reducing the shopping cart abandonment rate. For example, the team could reduce the number of steps in the checkout process, they could provide the capability to save the shopping cart to make purchases later, or they could provide free shipping. Each of the proposed solutions is a "how" or "design" answer to the "what" or "requirement" to reduce the shopping cart abandonment rate. In addition, the original "what" of modifying the system to reduce the shopping cart abandonment rate might also be the "how" to the "what" of trying to improve sitewide conversion rates.

What versus how is a poor way to distinguish between requirements and design.

Determining Actual Business Need

Another common definition says that anything that defines the actual solution, such as the algorithms used, the look and feel, or the user interface elements, is design and does not belong with requirements. However, there are circumstances in which a particular request could sometimes be a requirement, and at other times be design. For example, in certain industries, a product must use a specific cryptographic algorithm to be competitive; therefore, it is a requirement. For a different application, the specific cryptographic algorithm might be completely irrelevant, and it is only important that the application encrypt credit card numbers using any algorithm.

The key difference between something that is a requirement and something that is not is whether the business stakeholders actually need it. You know that stakeholders don't actually need everything that they say they need, so your role is to determine whether a specific request actually is a requirement—whether they really need it or not.

Requirements Defined

A *requirement* is anything that the business needs to have implemented in the solution. Requirements, therefore, can include functional requirements, non-functional requirements, business rules, and even what many people traditionally call design. Instead of telling the business stakeholders what types of things they are allowed to specify, you can focus on serving them by helping them truly understand what they need—by using models.

This section provides some definitions of requirement terms that we will use throughout this book. A *functional requirement* is a behavior or capability that the solution can provide irrespective of any qualifiers. A *business rule* is a requirement that represents a conditional statement that modifies a functional requirement, including, but not limited to, when the function is available and who is allowed to execute the function. A business rule contains words such as "if," "when," and "then." A *non-functional requirement* is any requirement that is not a functional requirement (including business rules). A *feature* is a short-form description of an area of functionality that the solution will ultimately include to meet the business objectives. Features are collections of requirements that are used to articulate and organize the requirements. Table 1-1 shows a few examples.

TABLE 1-1 Requirements Examples

Requirement	Type
The system shall be able to automatically approve or deny credit.	Functional requirement
When the credit score is above 750, the system shall automatically approve credit.	Business rule
The system shall use the following algorithm when automatically determining credit approvals for scores less than 750: [*algorithm would be included here*]	Business rule
Approvals shall be returned to the user within 30 seconds.	Non-functional requirement

An *assumption* is a statement taken as truth, upon which decisions are made. Assumptions include any predictions or forecasts of the future. Assumptions are a topic critical to requirements because they are constantly made but rarely understood or articulated. In fact, when analysts are asked to write down their assumptions, they typically write down trivial ones that are not impactful, missing the important ones. These example assumptions might result in failure to achieve the business objectives if they prove to be incorrect:

- Many people are willing to search online to resolve their technical issues.
- Fifty percent of people who are experiencing technical issues will be willing to wait for follow up.
- Ninety percent of the business's customers are online.
- The problems to be resolved can be solved by customers on their own.

Requirements Models Are Not the End Game

Using requirements models does not eliminate the need to write requirement statements. The models provide context and create a full picture of the requirements, but they do not represent the final requirements that will be used by the system developers and testers, so you will need to take additional steps to derive requirements from the models. Like a grocery list that is organized by aisle, the requirements form the checklist for the team to develop the solution. The value of models is in helping organize the requirements in a way that makes it easy to see when requirements are missing, extraneous, or incorrect.

You should include all the models you create as part of the complete set of requirements artifacts on a project. However, only the combination of textual and visual requirements can paint a full picture of the solution to be built (Wiegers 2003).

Using RML on Projects

You can think of the RML models described in this book as a toolbox of models and templates for use on software projects. Multiple models should generally be used together, and there are common approaches that define when to use specific models in the overall development life cycle. The approaches for applying requirements models on projects work with many development methods, such as agile, iterative, and waterfall methods (see Chapter 25, "Selecting Models for a Project").

Additional Resources

- "RML Quick Reference for Business Analysts" is a two-page summary of the models: *http://www.seilevel.com/wp-content/uploads/RML-Language-for-Modeling-Software-Requirements.pdf*.
- Karl Wiegers provides a solid introduction to the value of models in Chapter 11 of *Software Requirements, Second Edition* (Wiegers 2003).

References

- Chen, Anthony. 2010. "What vs. How – BRD vs. User Requirements vs. Functional Requirements": *http://requirements.seilevel.com/blog/2010/04/what-vs-how-brd-vs-user-requirements-vs-functional-requirements.html*.
- Cowan, Nelson. 2001. "The Magical Number 4 in Short-Term Memory: A Reconsideration of Mental Storage Capacity." *Behavioral and Brain Sciences* 24, 87-114.
- Ellis, Keith. 2008. "Business Analysis Benchmark Report." IAG Consulting. *http://www.iag.biz/images/resources/iag%20business%20analysis%20benchmark%20-%20executive%20summary.pdf*.
- Gottesdiener, Ellen. 2002. *Requirements by Collaboration: Workshops for Defining Needs*. Boston, MA: Addison-Wesley Professional.
- Miller, George A. 1956. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information." *Psychological Review* 63, 81-97.
- Object Management Group. 2007. "OMG Unified Modeling Language Specification." *http://www.uml.org/#UML2.0*.
- The Standish Group. 2009. "CHAOS Summary 2009." West Yarmouth, MA: The Standish Group International, Inc.
- Wiegers, Karl E. 2003. *Software Requirements, Second Edition*. Redmond, WA: Microsoft Press.

CHAPTER 2

Model Categorization

Imagine that you need to cut a round hole in a sheet of plywood. You have a shelf full of tools that you could use. You would very quickly narrow your search to those tools that can cut. For example, you would immediately pass over the hammer, the file, and the screwdriver. But you would zero in on all the different types of cutting tools such as scissors, tin snips, drills, routers, ripsaws, jigsaws, and handsaws. From there, you would focus on selecting the one that could make the circular cuts you need in the easiest way possible. Some of the tools might require more setup because they use an air compressor instead of a battery or electrical outlet. The point is that you have "categorized" your tools by type and purpose.

You can apply the same concept of categorization to requirements models to help you select models for specific types of analysis. RML models are organized into categories of objectives models, people models, systems models, and data models, known collectively as OPSD. The RML classification, represented by the diagram in Figure 2-1, offers a complete toolbox of models for analyzing the solution to be built. The RML models together allow you to look at the objectives of the solution, the people who are using the solution, the systems themselves, and the data that is being processed. These models bound the analysis to provide you with the best possible chance to ensure that you don't miss key requirements and that you avoid including unnecessary requirements.

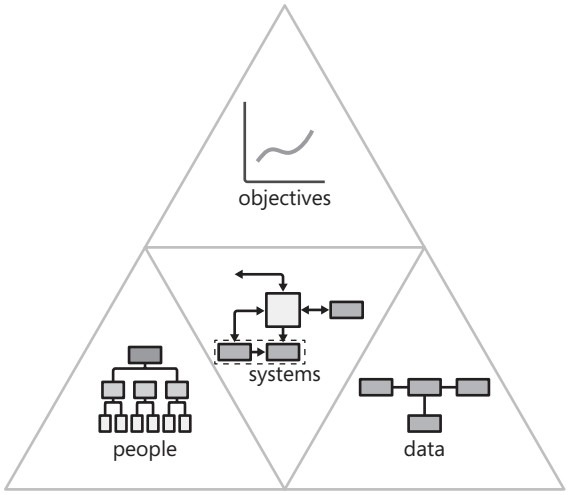


FIGURE 2-1 The OPSD classification of RML models.

Objectives, People, Systems, and Data Models

Software has a single purpose: to process data. Simply put, data enters the system, is processed, and then exits the system. The earliest models for software development, such as flow charts and structured design, took this design-centric view (DeMarco 1978). This view also considered how various systems in a multisystem environment transferred data to each other. Over the past 25 years, solution teams have discovered that another view is vitally important—that of the end user. Out of that discovery rose Use Cases, business process modeling, and other forms of user-centric modeling. Finally and most recently, executive stakeholders have been trying to determine how to align software development with end-user and organizational business objectives.

RML's organizational structure is based on these traditional model areas and groups models by objectives, people, systems, and data (OPSD). These areas represent four categories of information that you need to consider to thoroughly analyze your solution. RML models help you create boundaries around your solution by helping you start with information that you can easily ensure is complete.

Table 2-1 shows the RML models organized by OPSD.

TABLE 2-1 RML Model Categorization

	Description	Models	Bounding Model
Objectives	Describe the business value of the system and help you prioritize features and requirements based on their value	Business Objectives Model Objective Chain Key Performance Indicator Model Feature Tree Requirements Mapping Matrix	A Business Objectives Model bounds the objectives space
People	Describe who is using the system, along with their business processes and goals	Org Chart Process Flow Use Case Roles and Permissions Matrix	An Org Chart bounds the people space
Systems	Describe what systems exist, what the user interface looks like, how the systems interact, and how they behave	Ecosystem Map System Flow User Interface Flow Display-Action-Response Decision Table Decision Tree System Interface Table	An Ecosystem Map bounds the systems space
Data	Describe the relationships between business data objects from an end-user perspective, the life cycle of the data, and how that data is used in reports to make decisions	Business Data Diagram Data Flow Diagram Data Dictionary State Table State Diagram Report Table	A Business Data Diagram bounds the data space