# Good Practices for Developing User Requirements©

Ellen Gottesdiener
*EBG Consulting*

*Defining user requirements – the needs of the stakeholders who directly interact with the system – is arguably one of the most difficult challenges in building complex systems. When it comes to defining user requirements for software, it is essential to use models to document and analyze the requirements. This article provides a requirements model roadmap that helps software development teams understand the effective use of requirements models. It also describes good practices for creating and using these models.*

Many software developers have a love-hate relationship with requirements. They love having a list of things they need to engineer into the product they are building, but they hate it when the requirements are unclear, inaccurate, self-contradictory, or incomplete. They are right to be concerned.

The price is high for teams that fail to define requirements or that do it poorly. Ill-defined requirements result in requirements defects, and the consequences of these defects are ugly [1- 6]:

- Expensive rework and cost overruns.
- A poor quality product.
- Late delivery.
- Dissatisfied customers.
- Exhausted and demoralized team members.

To reduce the risk of software project failure and the costs associated with defective requirements, project teams must address requirements early in software development and they must define requirements properly.

## A Short Review of Requirements

Before we get to the nitty-gritty of building requirements models, let us look at some basic requirements concepts. User requirements – the focus of this article – are one of three types of requirements (see Figure 1). The other two types are those related to the mission or business and those that describe the software itself.

*Business requirements* are statements of the business rationale for the project. These requirements grow out of the vision for the product which, in turn, is driven by mission (or business) goals and objectives. The product's vision statement articulates a long-term view of what the product will accomplish for its users. It should include a statement of scope to clarify which capabilities are and are not to be provided by the product.

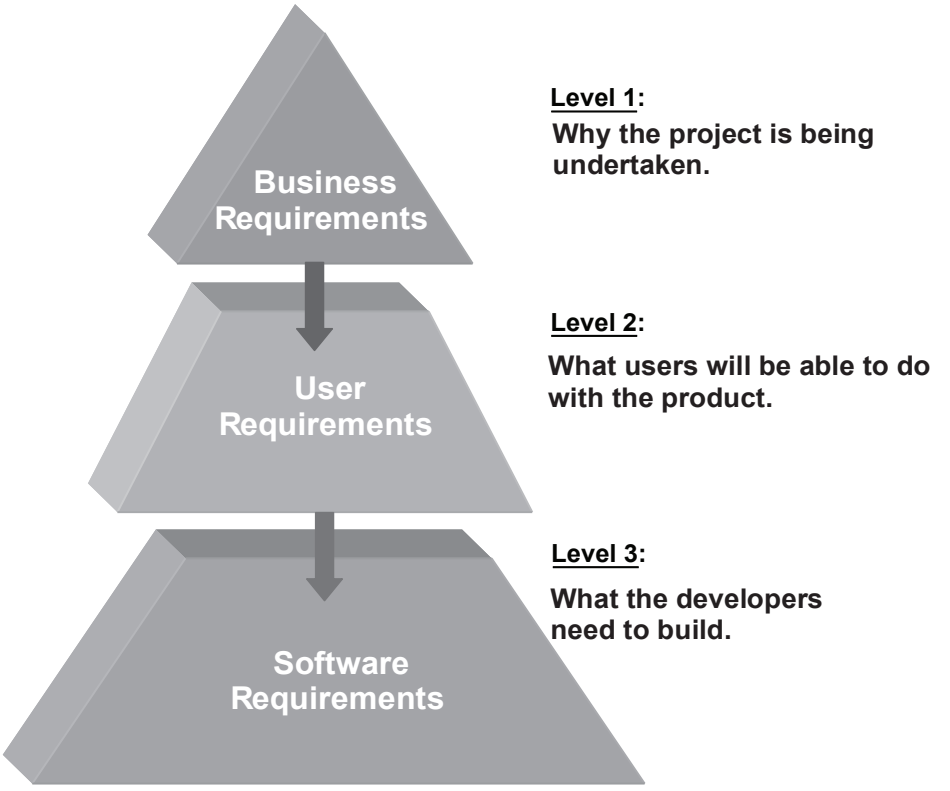
*User requirements* define the software requirements from the user's point of view, describing the tasks users need to accomplish with the product and the quality requirements of the software from the user's point of view. *Users* can be broadly defined to include not only the people who access the system but also inanimate *users* such as hardware devices, databases, and other systems. In the systems produced by most government organizations, user requirements are articulated in their concept of operations document.

*Software requirements* are detailed descriptions of all the functional and nonfunctional requirements the software must fulfill to meet business and user needs. Nonfunctional requirements include software design constraints, external interfaces, and quality attributes such as performance, security, installation ability, availability, safety, reusability, and more [7]. Software requirements, which are documented in a software requirements specification, establish an agreement between technical specialists and business managers on what the product must do.

The key activities in requirements development are the following: *elicitation*, *analysis*, *specification*, and *validation* [8]. In *elicitation*, you identify the sources of requirements and solicit requirements from those sources. Requirements elicitation relies on appropriate stakeholder involvement, one of the most critical elements for project success [9]. The goal of requirements *analysis* is to sufficiently understand and define the requirements so that stakeholders can prioritize and allocate them to software. *Specification* involves differentiating and documenting functional and nonfunctional requirements and checking that the requirements are documented unambiguously and completely. *Validation* examines the requirements to ensure that they satisfy user's needs.

Elicitation and analysis are crucial early

Figure 1: *Requirements Levels*



activities that require intense stakeholder involvement. To analyze the requirements you are eliciting, a key good practice is to create *requirements models* (also referred to as *analysis models*): user requirements represented by text (such as tables, lists, or matrices), diagrams, or a combination of text and graphical material [7]. These models facilitate communications about requirements with your stakeholders.

As you elicit requirements from stakeholders and represent them using requirements models, you should verify your models to ensure they are internally consistent. You also need to prioritize your requirements: With active user involvement, you analyze the trade-offs among requirements to establish their relative importance [8].

## The User Requirements Model Roadmap

Now let us take a closer look at user requirements models. The beauty of the Requirements Model Road Map (Figure 2) is that it shows the relationships between the three types of requirements (business, user, and software) and categorizes the models you can use to represent each type. Each model is designed to answer one of the *5Ws + 1H* questions: *Who? What? When? Why? How?* [7].

(Note that the question *Where?* provides information mainly about nonfunctional requirements. Although these are not user requirements – which depict functional requirements – analysts asking *Where?* during analysis will also discover a slice of useful quality attributes such as performance and usability).

In addition, the user requirements model falls into three categories: scope, high-level and detailed, and alternative models. Some models (shown in italics in Figure 2) are useful for analyzing the business process, and others are useful for clarifying project scope. Defining stakeholder categories early in elicitation, for example, identifies the people you should involve in requirements modeling. High-level and detailed models, such as use cases, a data model, and business rules, can reveal requirements defects such as errors, omissions, and conflicts. Requirements analysts and engineers can substitute alternative models when the engineers better communicate requirements or fit the project culture.

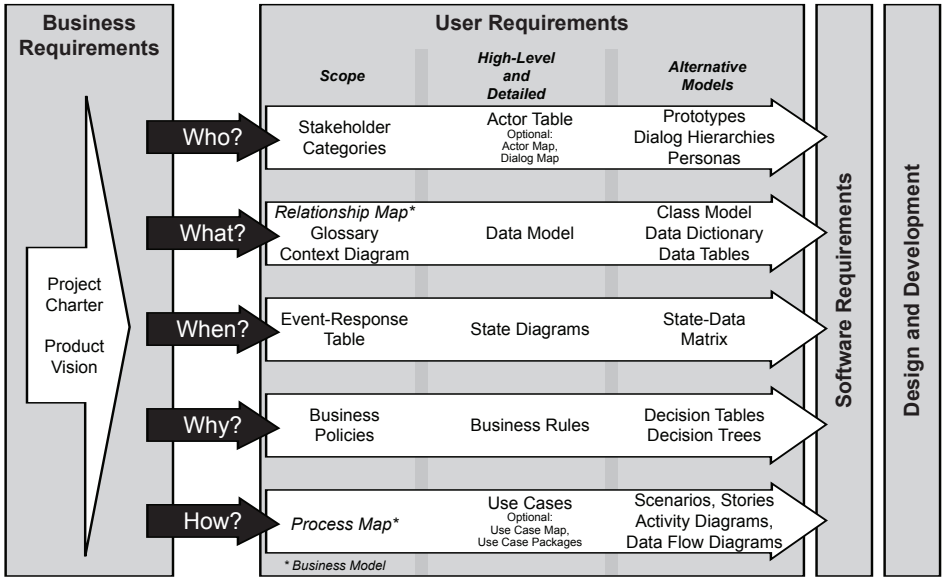
Each requirements model represents information at a different level of abstraction. A model such as a state diagram represents information at a high level of abstraction, whereas detailed textual requirements represent a low level of abstraction. By stepping back from the trees (textual requirements) to look at the forest (a state diagram), the team can discover requirements defects not evident when reviewing textual requirements alone.

Because the requirements models are related, developing one model often leads to deriving another. Examples of one model driving another model are the following:

- Actors initiate use cases.
- Scenarios exemplify instances of use cases.
- A use case acts upon data depicted in the data model.
- A use case is governed by business rules.
- Events trigger use cases.

In this way, you can use various routes to harvest one model from another. This approach helps you develop the models quickly while at the same time verifying the model's completeness and correctness.

Figure 2: *User Requirements Model Roadmap*




## User Requirements Models

Here, in alphabetical order, are brief descriptions of the common user requirements models shown in the User Requirements Models Road Map.

### *Activity Diagram*

An activity diagram is a model that illustrates the flow of complex use cases using Unified Modeling Language (UML) notation. This model is useful for showing use case steps that have multiple extension steps, and for visualizing use cases.

### *Actor Map*

An actor map is a model that shows actor interrelationships. An actor map supplements the actor table and can also be used as a starting point for identifying use cases. Actors can be written on index cards (one per index card) or drawn using the UML notation. UML depicts actors in an actor map as stick figures, as boxes (supplemented by the notation "<<Actor>>"), or as a combination (e.g., stick figures for human actors, and boxes for nonhuman actors).

### *Actor Table*

An actor table is a model that identifies and classifies system users in terms of their roles and responsibilities. This model helps reveal missing system users and identifies functional requirements as user goals (use cases), and also management to clarify job responsibilities.

### *Business Policies*

Business policies are guidelines, standards, and regulations that guide or constrain the conduct of a business. Policies are the basis for the decision making and knowledge that are implemented in the software and in manual processes. Whether imposed by an outside agency or from within the company, policies are used to streamline operations, increase customer satisfaction and loyalty, reduce risk, improve revenue, and adhere to legal requirements. This model helps you identify policies allocated to business people, which in turn allows management to prepare for software implementation by updating procedures, guidelines, training, forms, and other assets needed to enforce the policies. Some policies are also allocated to software for implementation.

### *Business Rules*

Business rules are text statements that decompose business policies. Business rules describe what defines, constrains, or enables the software behavior. You use

business rules to specify the controls that govern user requirements and to clarify which rules should be enforced in software and which will be allocated to business people. Because business rules require data, defining the rules will uncover needed data. User requirements depend on the complete and correct enforcement of business rules.

### Class Model

A class model is a diagram that shows the classes to be used in a system. A *class* is the generic definition of a collection of similar objects (persons, places, events, and physical artifacts). You use a class model in projects employing object-oriented software development methods, tools, or databases.

### Context Diagram

A context diagram is a model that shows the system in its environment with the external entities (people and systems) that provide and receive information or materials to and from the system. This model helps stakeholders to quickly and simply define the project's scope and to focus on what the system needs as inputs and provides as outputs. A context diagram helps the team derive requirements models (such as actors, use cases, and data model information) and can reveal possible scope creep problems as new external entities are added.

### Data Dictionary

A data dictionary is a model that provides a description of the data attributes and structures used in a system. This model is a central place for defining each data element and describing its data type, length, and format. Some project teams use data modeling tools that provide data dictionary capabilities.

### Data Flow Diagram (DFD)

A DFD is a model that shows related inputs, processes, and outputs for processes that respond to external or temporal events. Unlike use cases (which are oriented toward actor goals), DFDs focus on the data that goes in and out of each process, taking an internal view of how the system handles events.

### Data Model

A data model shows the informational needs of a system by illustrating the logical structure of data independent of the data design or data storage mechanism. You use a data model to identify, summarize, and formalize the data attributes and structures needed to satisfy functional requirements and to create an easy-to-maintain database. Data models help to simplify design and programming and help identify external data entities (other systems that supply data to the software).

### Data Table

A data table is a model in the form of a table that contains sample data to elicit and validate a data model or data dictionary. Each row represents a set of occurrences in an entity, and each column represents sample attributes.

### Decision Table

A decision table is a model that specifies complex business rules concisely in an easy-to-read tabular format. A decision table documents all the possible conditions and actions that need to be accounted for in business rules. *Conditions* are factors, data attributes, or sets of attributes and are equivalent to the left side of atomic business rules. *Actions* are conclusions, decisions, or tasks and are equivalent to the right side of atomic business rules. Factors that must be evaluated form the top rows of the table. Actions make up the bottom rows of the table.

### Decision Tree

A graphical alternative to a decision table, a decision tree presents conditions and actions in sequence. Each condition is graphed with a decision symbol representing *yes* or *no* (or a *true* or *false* conclusion). Branches to additional conditions are drawn left to right. Actions are drawn inside rectangles to the right of the branch to which they apply.

### Dialog Hierarchy

A dialog hierarchy is a model that shows the dialogs in a system (or Web page) as a hierarchy. It does not show transitions.

### Dialog Map

A dialog map is a diagram that illustrates the architecture of a system's user interface. It shows the visual elements that users manipulate to step through tasks when interacting with the system. Dialog maps can be used to uncover missing or erroneous use case paths and to validate use cases, scenarios, or both in requirements walkthroughs with users.

### Event-Response Table

An event-response table model identifies each event (an input stimulus that triggers the system to carry out a function) and the event responses resulting from those functions. An event-response table defines the conditions to which the system must respond, thereby defining the functional requirements at a scope level. (Each event requires a predictable response from the system.) This model can also reveal needs for external database access or file feeds.

### Glossary

The glossary is a list of definitions of business terms and concepts relevant to the software being developed or enhanced.

### Persona

The persona is a description of an actor as a fictitious system user or archetype. You describe each persona as if he or she is a real person with personality, family, work background, preferences, behavior patterns, and personal attitudes. The focus is on behavior patterns rather than job descriptions. Each persona description is written as a narrative flow of the person's day with added details about personality. Four or five personas represent the roles that use the system most often or are most important to the functional requirements.

### Process Map

A process map is a diagram that shows the sequence of steps, inputs, and outputs needed to handle a business process across multiple functions, organizations, or job roles. This model helps you identify the processes that are allocated to the business (manual processes) and those that will be allocated to software.

### Prototype

A prototype is a partial or preliminary version of a system created to explore or validate requirements. Exploratory prototypes can be mock-ups using paper, whiteboards, or software tools.

### Relationship Map

A relationship map is a diagram that shows the information and products that are exchanged among external customers, suppliers, and key functions in the organization. This model helps you understand the organizational context of the project by identifying affected business functions and their inputs and outputs.

### Scenario

A scenario is a description of a specific occurrence of a path through a use case (i.e., a use case instance). Example: A customer calls to reschedule a job, adding another service and requesting a repeat customer discount.

### Stakeholder Categories

Stakeholder categories are structured

arrangements of groups or individuals who have a vested interest in the product being developed. You use this model to understand who has an interest in or who has influence on the product, who will use the software and its outputs, and who the product will affect in some way. These groups and individuals will need to be kept informed about requirements progress, conflicts, changes, and priorities.

### *State-Data Matrix*

A state-data matrix model shows attributes that are added or changed during a state change. Each attribute is identified in the data model and data dictionary.

### *State Diagram*

A state diagram is a visual representation of the life cycle of a data entity. Events trigger changes in data, resulting in a new state for that entity. Each state is a defined condition of an entity, a hardware component, or the entire system that requires data, rules, and actions. A state diagram can also show actions that occur in response to state changes. You use a state diagram to understand how events affect data and to identify missing requirements such as events, business rules, data attributes, and use case steps.

### *Story*

A story is a text description of a path through a use case that users typically document. Stories replace use cases and scenarios when you are planning releases for agile projects. Stories are essentially detailed scenarios, but each story is judged by developers to require less than two weeks to develop. When combined with acceptance tests, stories are roughly equivalent to use cases.

### *Use Case*

The use case describes in abstract terms how actors use the system to accomplish goals. Each use case is a logical piece of user functionality that can be initiated by an actor and described from the actor's point of view in a technology-neutral manner. Use cases summarize a set of related scenarios. The purpose of use cases is to reveal functional requirements by clarifying what users need to accomplish when interacting with the system. Use cases are a natural way to organize functional requirements and can be easier for users to understand and verify than textual functional requirements statements.

### *Use Case Map*

The use case map is a model that illustrates the work flow of use cases. Each use case map represents a set of highly cohesive use cases sharing the same data, often triggered by the same events or initiated by the same actor.

### *Use Case Package*

The use case package is a logical, cohesive group of use cases that represents higher level system functionality. You create a use case package by combining use case maps or grouping use cases. Most systems will have multiple packages. You can use a UML file folder notation to show each package, and you can name each package according to its functionality.

## Good Practices for Modeling User Requirements

Following good requirements modeling practices (see *Good Practices for Modeling User Requirements*, Table 1) is the key to successful development of user requirements. These practices accelerate modeling, engage stakeholders, and give you high-quality requirements – ones that are correct, complete, clear, consistent, and relevant.

The first good practice is to represent and agree on the project's scope early in requirements elicitation. Why? It has to do with scope creep – the unrestrained expansion of requirements as the project proceeds. Scope creep is one of the greatest risks in software development [6]. A clear definition of product scope narrows the project's focus to enable better planning, better use of time, and better use of resources. Moreover, scope-level models establish a common language that team members can use to communicate about the requirements and help to articulate the boundary between what is in and what is not in scope for the product.

Another good practice, as mentioned earlier, is to document your product using multiple user requirements models. Each model describes one aspect of a problem the product will address. Thus, no single model can describe all the requirements. Furthermore, elements of one model often link to elements of another, so one model can be used to uncover related or missing elements in another model.

It is also good to use both text and graphics to represent user needs. Multiple representations tap into different modes of human thinking. Some people think more precisely with words, and others understand concepts more quickly via diagrams. Using both types of representations leverages these different thinking modes. In addition, mixing text and graphics makes requirements development more interesting and engaging. It provides variety and permits stakeholders to understand their requirements from more than one angle.

You should also select models that fit the domain of your product. That is because some models are better suited to communicate requirements for certain domains. For example, *When* models (such as an event-response table and a state machine diagram) are well suited to dynamic domains – those that respond to continually changing events to store data and act on it based on its state at a point.

Another well-known good practice is to develop your requirements iteratively. Each iteration is a self-contained mini-project in which you undertake a set of activities – elicitation, analysis, specification, and validation – resulting in a subset of requirements. The rationale for this practice is that user requirements seldom remain unchanged for a long period. On teams using agile methods, each iteration also incorporates the work needed to deliver the working software that satisfies those requirements. In some domains, requirements change faster than the system or subsystem can be developed. In addition, the cost of implementing changes increases dramatically as the project proceeds. Developing requirements in an evolving manner is essential in reducing these risks.

You can also use requirements models to identify requirements defects. The interconnections among the models help to expose any inconsistencies in related models. This self-checking accelerates the team's ability to uncover missing, erroneous, vague, or conflicting requirements.

Table 1: *Summary: Good Practices for Modeling User Requirements*

| | |
|---|---|
| 1. | Define, represent, and agree on the project's scope early in requirements elicitation. |
| 2. | Document your product using multiple user requirements models. |
| 3. | Select models that fit the domain of your system. |
| 4. | Develop requirements models iteratively. |
| 5. | Use requirements models to identify requirements defects. |
| 6. | Use models to communicate: Create simple, readable diagrams focused less on beauty and more on understanding. |
| 7. | Conduct retrospectives as you iterate through requirements development. |

When you are creating graphical models, it is crucial to create simple, readable diagrams. The benefit of diagrams is that they give you a way to quickly communicate complex, controversial, or unclear requirements. Thus, you should avoid complex, hard-to-read diagrams. Draw diagrams manually to begin with or use an easy-to-learn drawing tool. Keep them simple and easy to read. Focus on maintaining accuracy and exposing unclear or incorrect requirements – not beauty or completeness.

The final good practice I want to mention applies whether or not you are using modeling: I always tell my clients to conduct short retrospectives at the end of each requirements iteration. A *retrospective* is a special meeting in which the team explores what works, what does not work, what can be learned from the just completed iteration, and what ways to adapt their processes and techniques before starting another iteration [10, 11]. Retrospectives allow for early learning and correction and may be your team's most powerful tool for process improvement.

## On Your Way

Software development teams enjoy access to a world of tools and technologies, but building truly successful software still depends on team members gaining a deep understanding of user needs. When your team is developing a software product, you will save time, money, and frustration by using appropriate models to describe and analyze the product's user requirements.◆

## References


1. Reifer, Donald J. "Profiles of Level 5 CMMI Organizations." CROSSTALK Jan. 2007.
2. Schwaber, Carey. "The Root of the Problem: Poor Requirements." IT View Research Document. Forrester Research, 2006
3. Dabney, James B., and Gary Barber. "Direct Return on Investment of Software Independent Verification and Validation: Methodology and Initial Case Studies." Assurance Technology Symposium, June 2003 <http://sarpresults.ivv.nasa.gov/ViewResearch/24.jsp>.
4. Hooks, Ivy F., and Kristina A. Farry. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. New York: Amacom, 2001.
5. Nelson, Mike, James Clark, and Martha Ann Spurlock. "Curing the Software Requirements and Cost Estimating Blues." The Defense Acquisition University Program Manager Magazine Nov.-Dec. 1999.
6. Jones, Capers. Patterns of Software Systems Failure and Success. Boston, MA: Thomson Computer Press, 1996.
7. Gottesdiener, Ellen. Software Requirements Memory Jogger: A Pocket Guide to Help Software and Business Teams Develop and Manage Requirements. Methuen, MA: Goal/QPC, 2005.
8. Institute of Electrical & Electronics Engineers (IEEE). "IEEE Software Engineering Body of Knowledge." IEEE Computer Society, 2004 <www.swebok.org>.
9. Standish Group International. "CHAOS Chronicles." Standish Group International, 2003.
10. Kerth, Norman L. "Project Retrospectives: A Handbook for Team Reviews." New York: Dorset House, 2001.
11. Gottesdiener, Ellen. "Team Retrospectives for Better Iteration Assessment." The Rational Edge Apr. 2003 <http://ebgconsulting.com/Pubs/Articles/TeamRetrospectives-Gottesdiener.pdf>.


## About the Author


**Ellen Gottesdiener**, principal consultant, EBG Consulting, helps get the right requirements so projects start smart and deliver the right product at the right time. Her book, "Requirements by Collaboration: Workshops for Defining Needs" describes how to use multiple models to elicit requirements in collaborative workshops, and "The Software Requirements Memory Jogger" describes essentials for requirements development and management. In addition to providing training, eLearning and consulting services, she speaks at and advises for industry conferences, writes articles, and serves on the Expert Review Board of the International Institute of Business Analysis Business Analysis Body of Knowledge.

**EBG Consulting, Inc.**
**1424 Ironwood DR West**
**Carmel, IN 46033**
**Phone: (317) 844-3747**
**E-mail: ellen@ebgconsulting.com**