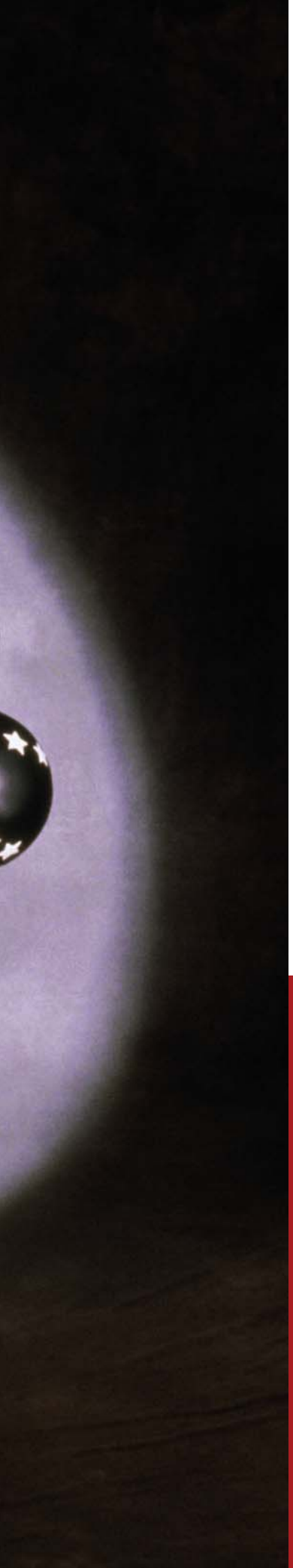

What Goes Up Must Come Down

By Jeff Patton

A Balanced Approach to Writing Requirements

"Well, this is almost right. I see some value in this workflow model you've created, but we need to rearrange this information into a more linear form so it tells a story that people can understand," the use case expert said, squinting at the user centered design (UCD) expert's task workflow model. "Our stakeholders can't confirm you're meeting their requirements with your model. That's why my use case model is so valuable—because it's easy to read and understand."

The UCD expert got a pained look on his face. "This workflow model is pretty easy to read. I can show you how to interpret it in a few minutes. But the real value here is that our stakeholders can see a concise picture of the whole system at a glance. Your use cases are forty pages of text—and getting longer daily. Our stakeholders can't understand all that detail. That's why my workflow model is so valuable—because it's easy to read and understand."

After a bit of discussion the two agreed that the workflow model could be restructured into several use cases, and that the use cases could be broken down and represented as a workflow model. "Well it's easy then," the use case expert grinned, "just rebuild your workflow model as a set of use cases and we'll be done." The UCD expert sighed, "I'll be too busy converting all these use cases into a workflow model I can understand!"

At first glance it may seem that these two are arguing over the best way to present the same information. And, to most people, there appears to be a lot of overlap among the different approaches to deriving requirements such as traditional functional decomposition, use cases, and user centered design. Both of these approaches produce requirements that can be given to designers and developers as the basis for their work. In an attempt to avoid these arguments, many organizations simply choose one approach and structure their requirements documents based on that choice. Because, after all, the results of all these approaches are about the same, right? Let's go back to our two experts and see.

The UCD expert has agreed to convert his workflow model into a few use cases to see how it goes. (Management seems to like the use cases better because there are more books on them, so the use cases must be better.) While reviewing the first use case derived from the UCD approach, the use case expert exclaimed, "Aha! You missed a key step here!" The UCD expert said, "Hmm . . . looks like you might be right, but what about those alternative steps I have here that you're missing?"

After much discussion, the use case expert and the UCD expert finally realized that the difference in format was just a distraction. The real difference was how the information was discovered. Working with use cases is generally a top-down approach while user centered design is usually bottom-up.

The Top and the Bottom▼

Let's review what we mean by "top" and "bottom." I like Alistair Cockburn's altitude metaphor in his book *Writing Effective Use Cases*. When describing use cases, Alistair discusses five levels:

- *Cloud Level*: very high summary, almost enterprise-level business goals
- *Kite Level*: summary, organizational-level goals
- *Sea Level*: user-level goals
- *Fish Level*: sub-function, short-term goal—part of meeting a sea-level goal
- *Clam Level*: too low, very detailed short-term goal

USE CASES

I often turn to Cockburn's book as a more detailed, approachable, and pragmatic way to write use cases. A use case describes the interaction between someone who has a goal and a system that helps achieve that goal—generally, but not always, a software system. Referred to as the "primary actor," this person might perform several tasks using the system in pursuit of the goal. The system is charged with the responsibility of helping the primary actor achieve that goal while preserving the interests of other stakeholders of the system.

Let's consider MegaDVD, a DVD retailer selling items in a brick-and-mortar retail store. In MegaDVD's phone order process, a primary actor is the Sales Associate. Other stakeholders are those within MegaDVD who want to make sure we accurately capture customer information and collect money.

For MegaDVD, a simple use case to take a phone order might look like Figure 1:

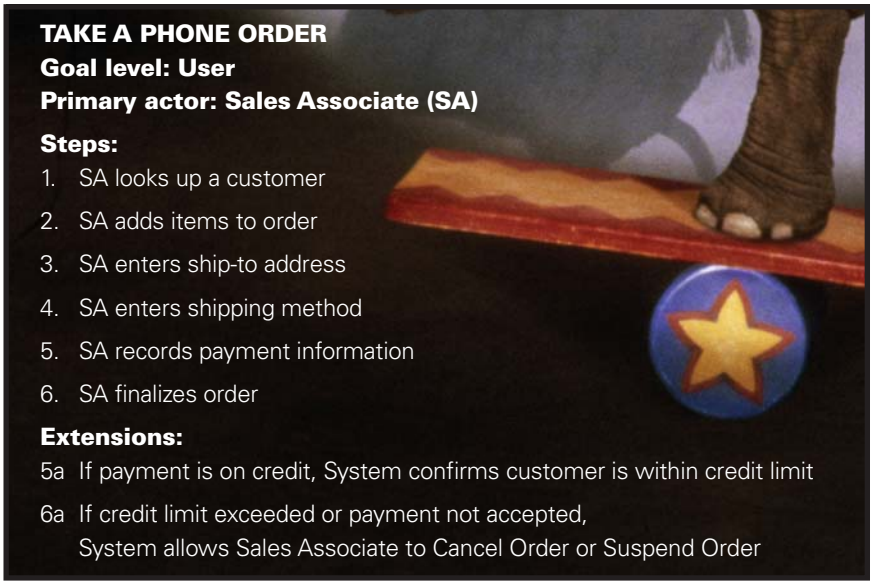

Figure 1: MegaDVD phone order use case.

Writing Use Cases Top-Down▼

If we were to begin documenting requirements for MegaDVD's sales process, we'd ask business leaders and domain experts about the people (actors) and the goals that the system should support. Next, we'd ask about the primary business processes the system should implement. This information is at the cloud and kite levels. (See Figure 2.) For MegaDVD, we'd discover that one of the kite-level concepts is the in-store sales process. With that knowledge, we might follow a path of functional decomposition noting how the sales process works as a whole, then moving down to how it works in an individual store, and finally, moving still further down to how an individual sale is rung up on the point of sale device. We're progressively moving down from high-level goals to lower-level goals and activities. We'll write

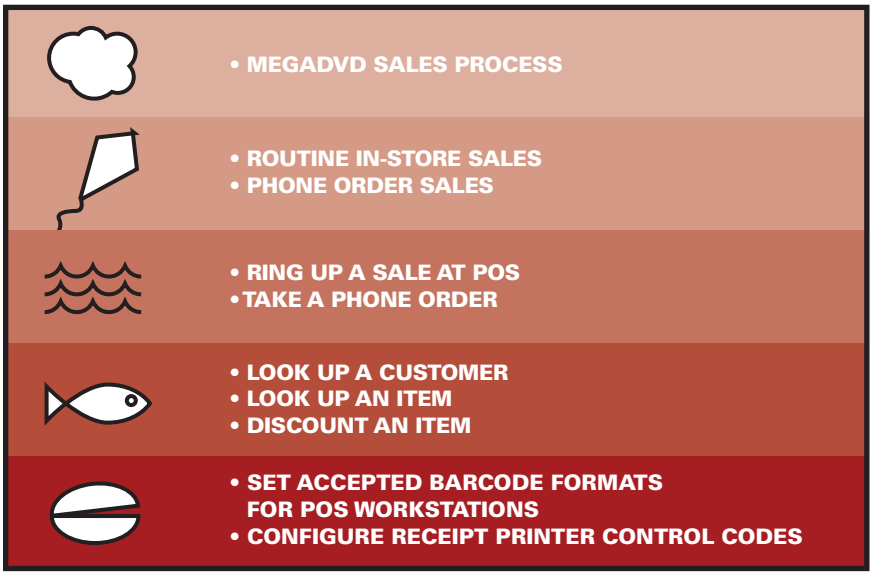

Figure 2: MegaDVD's use cases add detail as we descend. Notice that the closer we get to actually building the software, the more of these details we'll need to understand.

those low-level activities as numbered steps in our use cases.

Top-Down in Practice▼

While working top-down may seem straightforward, there are a few tricks to keep in mind:

Don't work exclusively top-down: If you're engaged in gathering and documenting requirements with use cases, you might have noticed that collaborative groups of participants seldom work purely top-down. In fact, the conversation about business process often starts in the middle by listing some sea-level use cases or business goals. Collect these use cases and then move up a level to write a use case at the summary, or cloud level. Once you've arrived at those cloud-level use cases, talking through them will help you spot sea-level use cases you might have missed.

While working through a use case, you'll recognize when the conversation changes goal level, either up or down. Use this as an opportunity to explore interesting details, or as a warning to pull the conversation back to the goal level you're currently exploring.

Focus on goals: Each use case should fulfill a user goal. The detailed steps are important—but only in support of a goal.

Consider exceptions and alternatives: When describing a business, it is important to ask at each step: "What could go wrong here?" "How would the system recover and meet the user's goals?" and "What might the user alternatively do at this step to reach his goal?" Capture these as extensions to your use case steps.

Know when to stop: As you break the functionality of the system down into ever-finer details, keep in mind the audience for your requirements. If you're in the early stages of a project where high-level scope and course-grain estimation are your goals, stop at sea level or earlier. Provide just enough detail to size the project and support the stakeholders of the system in evaluating the viability of the project. If you're in the later stages of the project, more detail will be necessary to support developers' detailed design and implementation.

USER CENTERED DESIGN

The term "user centered design" was shortened from Norman and Draper's *User Centered System Design*, published in 1986. This book described the growing trend in software design to focus first on the individual users of the system, their behaviors and goals, and then design functionality in response to that understanding. Over time, specific UCD approaches have been published. Popular approaches include Constantine and Lockwood's usage-centered design, Cooper's goal-directed design, Holtzblatt and Beyer's contextual design, and Carroll and Rosson's scenario-based design. While these authors offer slightly different approaches to designing software, they generally agree that it's best to first gather lots of detailed information about your users from a variety of sources. Most common sources are user interviews and observation of users "in the wild." Interview and observation notes usually contain large amounts of detailed information—clam- and fish-level data. Since UCD approaches generally rely on this type of foundational data, I'll classify these as bottom-up approaches.

Bottom-up in Practice▼

Suppose we want to gather requirements for MegaDVD's new system. We'll start by scheduling some visits to MegaDVD stores. Rather than beginning with interviews, we'll observe users doing their work. We'll watch people ring up sales and take orders over the phone. We'll take note of their environment, carefully observing how users really do their jobs. As important clues about employee activities, we'll look for things like Post-It notes with most recently used customer numbers or UPS and FedEx cost charts for quoting shipping charges to phone customers. These details about the physical space in which people work often are not identified in the top-down approach. Later, we'll schedule interviews with these users and other stakeholders to ask them to describe their business processes.

While working bottom-up, there are a few tricks to keep in mind:

While observing and interviewing, take detailed notes. The most efficient way to gather details is during on-site visits and user interviews. Take lots of notes. Try to keep one idea per line. Each idea is a low-level detail. Later, we'll examine all these details to find patterns. Separating them in your notes now will make it easier later.

Focus on concrete details. Using Cockburn's altitude metaphor, try to capture clam-level details and don't abstract or generalize—there will be time for that later. When interviewing users, you'll notice they often describe a business process by saying, "Well generally I..." If you hear that, stop them and ask them specifically what they did the last time

they performed this business process. Record those facts. If you do record information that is a generalization or interpretation, label it as such.

A picture is worth a thousand details. When talking with users on-site, take photographs of their workplaces. Note little details like the Post-It notes on their desk and the amount of work in their inbox. Take pictures, draw diagrams, and make copies of forms, notes, or anything else that seems important.

Don't trust your memory—transcribe notes immediately. After a day on-site, you will have many pages of notes, drawings, and photographs. Immediately transcribe these hastily written notes. Transcribing notes soon after their collection helps you fill in details you might have missed when writing quickly. I prefer recording notes in a spreadsheet, one row per idea. Placing them in an electronic form will help you later to categorize notes and print them in a form that's useful for our next step: building an affinity diagram.

Distill the details with affinity diagrams. After a couple of site visits and some interviews, you'll be swimming in details. The fastest way to distill these into usable information is to construct an affinity diagram. An affinity diagram is a great way to help common themes and concepts emerge from a large amount of data. (See the StickyNotes for a look at a finished affinity diagram.)

To build an affinity diagram from your notes, follow these steps:

1. Gather a team of people familiar with the business problem.
2. Transcribe each note onto a note card or Post-It note.
3. Cover a worktable with large sheets of poster paper.
4. Divide the notes among team members.
5. Place notes on the work surface. Place similar notes together or overlapping, and place dissimilar notes far apart. Feel free to move notes around. Discussion and arguments are good. When you're finished, your notes should have formed lots of clumps or clusters. Attach the clustered cards to the paper with tape.
6. As a group, talk through each cluster and then decide on a label for each cluster. Write that label on a different colored note card or directly on the poster paper. These labels are the distillation of the details you've captured.

Moving Up a Level from Details to User Tasks and Workflow▼

If you've followed a bottom-up process like the one described above, you have captured notes not only about the workflow that people are engaged in but also many other details about their environment, goals, and common problems. Now, with this understanding, it's easier to identify user tasks.While user task is a common term in the UCD community, I use the definition and approach offered by Constantine and Lockwood's usage-centered design—described in their book *Software for Use.*

A user task is anything a user physically does to achieve a goal. For example, I may have a goal to avoid tooth decay, and the tasks I perform daily to meet that goal include brushing and flossing. Technically I can have a goal to brush and floss, but let's call that a task since it describes something I do.

Good task names generally start with a verb and take the form "do something"—like "brush teeth," "look up customer," or "collect shipping information." I've found a good way to identify user tasks quickly is through brainstorming. After doing field observation and interviews, brainstorming is especially fruitful. You will find that many of the user tasks already are part of the affinity diagram.

A particular variation of brainstorming, popularized by Larry Constantine, is called "card-storming." Gather a diverse group of people around a worktable; the group that assembled the affinity diagram from the notes would be a good choice. Try to include users or others who have spent time with users in the wild. Have one or two team members volunteer to record tasks on 3x5 cards. Team members shout out ideas for tasks, and recorders write the tasks on cards and toss them into the middle of the worktable. Remember that the first rule of brainstorming is "no discussion or criticism." Say and record ideas. There will be time later for discussion and refinement. You'll know when brainstorming is finished because of the long silence. Wait a couple of minutes longer—until the silence starts to become unbearable. Once you've completed brainstorming, as a group go back through the task cards removing duplicates and poor task names. Take this opportunity to rewrite tasks that are unclear.

Now that you'll have a good set of user tasks, let's arrange them into a workflow model. As with affinity diagramming, cover the worktable with poster paper. Start arranging the tasks in chronological order—steps done first to the left of steps done subsequently. If some tasks might be done at the same time as others, place them directly below the most common task. When the steps appear in the correct order, use tape to affix them to the paper.

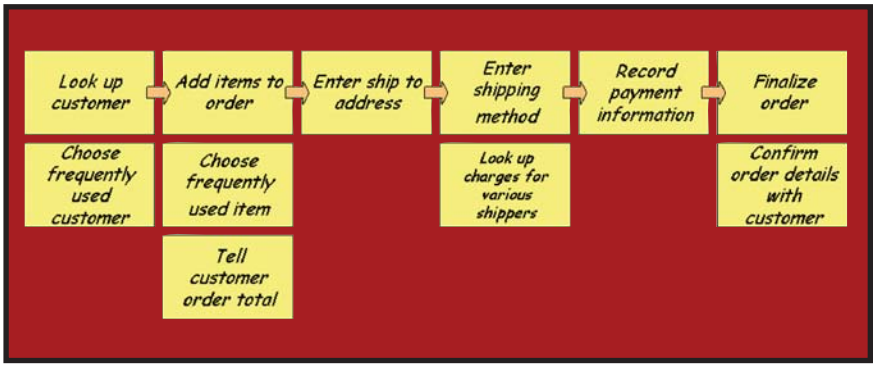

Figure 3: Workflow model of MegaDVD's phone order sales process.

Using the rules above for arranging tasks into a workflow, a model of MegaDVD's sales process for phone orders would look like Figure 3:

I read this model from left to right

and easily identify the possible workflows of a user taking a phone order. I arrived at this workflow bottom-up by observing users performing tasks in the field, compiling observed data, and then brainstorming the detailed tasks based on this observed data. It all came together (mostly) from the bottom-up.

above into a use case, we'll arrive at something like Figure 4:

The top-down approach results in the use case on the next column.

Comparing this with the use case we wrote thinking top-down, you'll notice a couple of interesting differences.

Through observation at the customer's site, we noticed that sales associates repeatedly took orders from the same customers and kept a list of most frequently accessed customers. They did the same with items. Consequently, from the bottom-up perspective we identified tasks for selecting frequently referenced customers and items.

We also noticed that customers didn't want to choose the shipping method without first comparing the costs of multiple shipping methods. This resulted in a task to look up shipping charges for different shippers. Hmm . . . we missed those things in our top-down use case.

But wait. When working top-down, we thought a bit more about exceptions and extensions—the special circumstances within a use case. The businesspeople in the room made a big point of this. This led us to develop use case steps to handle what happens when the customer's credit limit is exceeded, and we discovered situations where canceling and suspending orders was necessary.

TAKE A PHONE ORDER
Goal level: User
Primary actor: Sales Associate (SA)

Steps:

1. SA looks up a customer
2. SA adds items to order
3. SA enters ship-to address
4. SA enters shipping method
5. SA records payment information
6. SA finalizes order

Extensions:

1a SA chooses frequently used customer
2a SA chooses frequently used item
2b SA tells customer order total
3a SA looks up shipping charges for various shippers and tells customer
6a SA confirms order details with customer

Figure 4: Conversion of workflow model into a use case.

USE CASE AND TASK FLOW FACEOFF

We now find ourselves in the same place as our use case and UCD experts were at the beginning of our story. We've written use cases (mostly) top-down during requirements elicitation sessions with businesspeople, users, and domain experts. On the other hand, we've worked (mostly) bottom-up with field interviews, observations, and task workflows. Now it's time for these two approaches to battle it out.

Convert Information into a Common Format▼

The differences will begin to emerge as we convert our workflow models to use cases, and use cases to workflow models. Let's look at MegaDVD's phone order process as a use case. If we convert the task workflow model described

THE BEST SINGLE APPROACH ISN'T

We've approached this requirements work using two different techniques. Both helped us find interesting details. Is it possible that the best way to work is to use both top-down and bottom-up approaches simultaneously?

In truth, designers use both approaches. Hugh Beyer, co-author of *Contextual Design*, describes a bottom-up to top-down approach this way, " . . . when I teach, I often draw this as a mountain—climb the mountain on one side from little user data instances to general intents and themes; descend the mountain on the other side from high-level design solution to detailed specification."

Alternating Between High- and Low-Level Goals Yields the Best Results▼

Following a top-down approach, we start by establishing the high-level goals of our users, then the high-level business processes that support those goals. We continue to decompose from those goals down to specific activities users perform to achieve their goals, and we document those in the form of a use case.

Following a bottom-up approach, we start with details gathered in the field or from user interviews and then distill those details into user tasks and goals using affinity diagrams. Continuing up in our abstraction levels, we determine the kinds of users and the goals we observed them pursuing.

Curiously, at a high level, we often find that users or business stakeholders state the system's goals one way, but we observe that they pursue these goals differently in practice. Sometimes they even pursue entirely different goals. At a detailed level, we observe users doing things in an entirely different way than the policies and procedures state they should. Working top-down, we find that the specific activities users perform in practice may not be relevant to achieving the goals of the system as a whole. (See the

(Continued on page 50)

(Continued from page 32)
WHAT GOES UP MUST COME DOWN

StickyNotes for Raymonde Guindon's study of top-down and bottom-up design.)

AT THE END OF THE DAY

Although the finished requirements and their formats are interesting, it is the approach that leads to the requirements that has the greatest impact on their quality and, thus, the quality of the developed software. Working purely top-down or bottom-up is risky. Details are likely to be missed either way. You should consider requirements derived from only one approach to be lopsided—try to bolster those requirements by also using the other approach.

And finally, with all forms of requirements, the best validation is feedback from users of the finished software. If you can't wait that long to get feedback, develop prototypes early in your development process. If you can't build prototypes, collaboratively review requirements with your customers—from both the bottom and the top. **{end}**

Jeff Patton leads teams of Agile developers to build the best software possible. He proudly works at ThoughtWorks. Jeff's series of columns on software design and pre-design tips appears each quarter on StickyMinds.com.

Sticky Notes

For more on the following topics, go to www.StickyMinds.com/bettersoftware

- Example of a finished affinity diagram
- Raymonde Guindon study

(Continued from page 52)
YOUR JOB-REQUIREMENTS=LESS VALUE

analyst documents requirements, before the developer develops code, and before a tester tests the application, the skill of requirements analysis should stamp out most of the requirements issues that exist. All job functions should be able to effectively implement activities that will allow them to ensure requirements are complete, correct, consistent, non-ambiguous, verifiable, and traceable.

So, the last word that I have for this article is not a word but rather a final equation: **Tough love** plus **requirements mathematics** equals **Better Software.** **{end}**

A senior test consultant for Pointe Technology Group, Inc., Dion Johnson is responsible for providing IT consulting services that focus on the overall system development lifecycle, with particular focus on the quality assurance and quality control elements. He has presented at numerous SQE conferences and is a contributor to StickyMinds.com.