INFO TO GO

- Effectively communicating a design involves knowing what to say, how best to say it, and what to leave out.
- By choosing what to emphasize and using progressive realization techniques, you can unfold a design gradually, in successively more interesting parts.

Once upon a Design

TRANSLATING THE BEHAVIOR AND STRUCTURE OF YOUR SOFTWARE INTO COMPELLING STORIES

BY REBECCA WIRFS-BROCK

DESIGNING SOFTWARE, PLANNING A PROJECT, OR FIGURING out how to test complicated code can be a messy process. At times, you need to tidy things up and present your ideas to others. Francis Galton, a nineteenth-century geneticist, remarked, "It often happens that after being hard at work, and having arrived at results that are perfectly clear and satisfactory to myself, when I try to express them… I feel that I must begin by putting myself upon quite another intellectual plane. I have to translate my thoughts into a language that does not run very evenly with them." Similarly, if you want to effectively communicate your ideas, you need to translate them into compelling stories.

Identify Your Storyline

Before launching into a detailed presentation of your design, consider what you want to accomplish. What do you want to communicate? What's essential? What's less important? Perhaps you want to describe key collaborations between system components. Maybe you want to explain key aspects of your design to newcomers—the major subsystems, their responsibilities, and their patterns of interaction. Perhaps you want to introduce some central software objects and put them through their paces. Or maybe you want to describe how your design supports core use cases; and once people get the gist of these, you want to explain how your design handles exceptional situations. Perhaps you want to justify why you made certain decisions and get critical feedback.

If your story is comprehensive, there will be many things to say. Often, even a simple story has several key points. To start, list everything you want to discuss, whether it is big or small or it overlaps with something already on your list. List things you want to exclude, too.

Let's say you are designing a function that allows a bank account holder to make automatic payments from his account. Here are some key points about the design story you might want to cover:

- describe how major domain objects respond to a request to make a payment from a customer's account to a recognized vendor
- use a sequence diagram, but keep it simple
- point out calls to the backend banking system that could be bottlenecks
- start with a well-formed request (ignore UI details)
- relate the diagram to a "Make a Payment" use case (for which there are three alternatives)

Don't worry about how to organize your story or the items on your list until you have a large part of your story down. Even if your story is brief, you won't know the best way to present it until you've got the content in place and understand what your listeners or readers need to learn.

Understand Your Audience

If you are presenting your design to people just like you, consider yourself lucky. However, often your audience's interests and backgrounds differ. Some may know a lot and don't want to be bothered by hearing, yet again, things they already know. Others may need to understand design fundamentals before they can appreciate details. Some may want to know why you made the choices you did. They want to ask probing questions. After all, how can they write comprehensive tests if they don't know where the complexities lie? Still others may want only the punch line—what you ended up with—and don't want to hear about how you made the choices you did.

It's tough to please a diverse audience. You can't develop one story that will hold everyone's attention. You may even need to write several versions of the same story. More likely, you'll pick one intriguing path through all the many ways you could explain your design. Anticipate that, at times, some will want more details while others' interests will lag. Your goal should be to cater to your primary audience. Understand what they are likely to find interesting, confusing, difficult, or boring, and adjust your story accordingly. Be sure to balance their needs with the goals you have for telling your story.

While it is difficult to please everyone, a good storyteller isn't a defensive one. Assume that your audience wants to hear what you have to say and wants to follow your lead. If a significant fraction of your audience lacks basic information that they need to understand the details of your story, don't leave them behind. Prepare background material they can read beforehand. Or give them an overview in a separate presentation. Things that are only moderately interesting or are supplementary material for most folks can always be relegated to a separate handout.

Use Progressive Realization

There are ways to begin simply and then lead to more interesting or intricate views of your design. Landscape architects use the principle of progressive realization to design linked scenes. They create views that purposely conceal things. Points of interest are revealed only as you physically move through the landscape. Architects move people through the landscape in gradual, interesting steps. You, too, can set up your audience to comprehend things more deeply, a little bit at a time. As you move them through your design you can successively reveal more details or present different views of your design.

For example, when telling a high-level design story, stick to the main points. Tell it as if it were a news flash. Your audience will want to hear the highlights before deciding to invest more time in your story. Grab their attention! Present those things you want to emphasize first: What are the central objects or components? What is important about them? How do they collaborate? What's controversial or novel about your design approach? Reveal just enough to engage your audience. If you are writing a story, after setting the stage, give your readers options to veer off in one of several directions: to more detailed views, or perhaps to explore how your design handles exceptional conditions, or to reveal the inner workings of complex algorithms or more intricate collaborations. Offer different paths through your material. For each type of reader you can develop a customized roadmap that suggests an ordering of topics that they would find most appropriate.

Progressive realization works best when readers or listeners want to follow your lead. Those seeking specific facts won't sit still for long. They don't want to be led anywhere. They want to forge their own path. To counteract an impatient audience, you might prepare a section just for them that answers Frequently Asked Questions. Point them there while you stick to your storyline.

Stories build to more dramatic conclusions if important parts are told first, followed by new material presented in novel ways. For example, to describe how your software handles specific exceptional cases, start by defining the general strategies your software will follow. Then illustrate both a typical case and an exceptional condition with illustrative sequence diagrams. And instead of ending with many more mind-numbing diagrams, each illustrating how one particular exception is handled, conclude with a chart that summarizes each major exception, where it may occur, and its effect on the user.

Understand What's Fundamental

Fundamentals are the basic facts you need to know about your design story before other details make sense. If you know that most of your audience won't have the patience to wade through fundamentals, don't put the fundamentals first. If you want to reach out and present your design to other developers, testers, or architects for critical feedback, don't crowd your story with basic information that they can find out on their own. Building on fundamentals is most appropriate when your goal is to educate.

Even though you may not want to present design fundamentals first, it is important to know which aspects are more fundamental than others. This can help you anticipate questions or present your story in a more logical order. Here are some rules of thumb for gauging which fundamentals should be covered first.

Things you cannot change are more fundamental than things under your control. Descriptions of a problem are more fundamental than descriptions of a solution. For example, use cases are more fundamental than sequence diagrams that describe how a system supports them.

Things (software objects, concepts, physical entities) are more fundamental than the relations between them, their attributes, and their actions. Your audience needs to understand software objects and their general responsibilities before they can comprehend structural relationships or interaction details.

The normal case is more fundamental than the exceptional one. A "happy path" interaction sequence is more fundamental than an exception-filled one. If you want to explain both, you should separate the two.

Establish Scope, Depth, and Tone

The scope of your design story—how much territory you cover—depends on your goals. The depth of your story—how detailed your explanations are—depends on your reason for telling your story and the needs of your audience. Many stories have a narrow scope and limited depth. Often, stories are dashed off quickly to impart knowledge or get reactions. Their tone is informative, but brief. After all, you are around to answer questions.

Design stories can be more or less involved, depending on what needs to be said, how many details you describe, and how complex your software is. The further along you are, the more you know. So you can show and describe more aspects of your design—if that's appropriate.

For example, there are many ways to describe inter-object communications. Here are a few:

- a bird's-eye view of your system showing the overall architecture, major subsystems, and general paths of collaboration
- a view showing only the participants in a specific collaboration (and not showing any specific communications)
- a specific sequence of interactions among collaborators
- an in-depth view that shows and explains how objects interact under exceptional conditions
- a focused view that ignores some aspects in order to concentrate on a few, specific collaborators and their interactions
- an implementation view that explicitly describes precise message sequences
- a generalized view that illustrates how to adapt a particular collaboration

All these views have merit. You may present one or more of them in the same story, depending on what you need to communicate. Your design can be explained at many different levels. Your choice of level (or levels) should be based on how much you know and how much you want to reveal.

You can adjust the tone of your story to be more or less formal, authoritative, precise, comprehensive, and instructive. Diagram choices, as well as word choices, help set the tone. For example, UML sequence diagrams are more formal than UML collaboration diagrams. Both serve nearly identical purposes. When you want to throw out a rough design idea for comment, you don't want it to look too polished. Not every part of your story needs to be told in the same way or in the same depth. Formal and informal descriptions and diagrams all have their place. (For more on design presentation options, see this issue's StickyNotes at www.stqemagazine.com.)

Get comfortable describing your design at different levels of abstraction.

Give Elements Appropriate Emphasis

Things gain prominence by their position and appearance. This holds true for text and drawings. To increase an item's emphasis:

- put it first
- surround it with space
- put it in a bulleted list

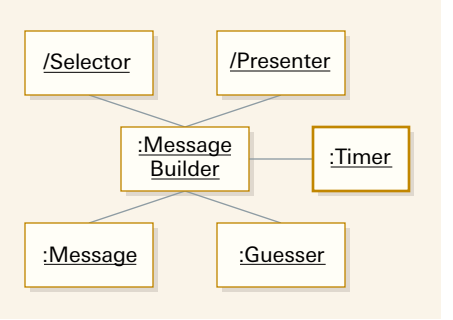

Figure 1: Placing the MessageBuilder object in the center draws attention to it.

- mention it in multiple places
- give it more room
- repeat or restate it in different forms

You can consciously attempt to emphasize or de-emphasize certain parts of your design story. Central location, size, and boldness add emphasis. Placing an object in the middle of a collaboration diagram gives it extra emphasis. UML active objects representing threads or processes are drawn in bold, giving those objects special emphasis. From the diagram in Figure 1, we see that when MessageBuilder, a controlling object, is placed in the center, it gets emphasized.

Certain things gain prominence by their position and size, whether you like it or not. Do your best to give design elements the appropriate emphasis. For example, many UML tools increase the size of a class to fit its name inside the box. This may inadvertently draw attention to unimportant classes with long names. I might go so far as to use a smaller font to display that lengthily named class, and place it in the lower right-hand corner. Sometimes emphasis will be misplaced, however, despite your best intentions.

Your choice of a use case template can emphasize certain elements. In Table 1,

TEMPLATE 1	TEMPLATE 2
Use Case: Make a Payment Author: Rebecca Last Revision Date: 7/11/03 Version: 0.4 Status: Preliminary Review Level: Summary . . .	Use Case: Make a Payment Actor: Bank Customer Pre-condition: User has an active account and is authorized to transfer funds . . . Author: Rebecca . . .

Table 1: Choose a template that emphasizes the things that are important about the use case.

the first template emphasizes version, author, and status. The second template features an actor and a pre-condition. Even though you may include authoring information, nothing says you have to place it first—unless that's what you want to emphasize.

Tell It, Draw It, Describe It: Some Guidelines

By now you've figured out your goals and your main storyline. You've analyzed your audience and understand what they need to know and at what level of detail. You also have some ideas on how to sequence your presentation. It's time to work on polishing your story. Countless writers have turned to Theodore Strunk and E.B. White's *The Elements of Style* for straightforward advice. Strunk and White's guidelines can be applied to design stories, too—whether they are written or drawn. Here's how you can apply the principles of good writing outlined in Strunk and White to design stories.

Do not overwrite. Ten pictures are not worth 10,000 words. Consider each drawing's purpose. If collaborations are similar, show a typical case first, then note how remaining ones differ. You can always draw a representative diagram that illustrates the typical case, and then explain all the interesting exceptions using a chart or table.

Be aware of monotony setting in. After seeing a bunch of nearly identical drawings, even the most stalwart designer's attention flags. To hold attention, shift their focus. For example, add commentary that explicitly calls out some details. Or point out that the next five diagrams are similar, so all but the most eager designer can skip them in good conscience.

Omit needless words. Stop short of telling everything. Keep explanations to the point. Avoid clutter when writing. Don't start a discussion with metatext—text that describes the text that follows. Don't pile on extra words or invent jargon; use simple language. Don't blindly fill in the blanks of a heavy-handed template.

You can also keep drawings simple without oversimplifying. Visual equivalents of "needless words" on a collaboration or sequence diagram include showing

- return values
- internal algorithmic details
- details of caching and lazy initialization
- object creation and destruction

Sometimes these details are important. If so, take exception to this guideline. Most of the time, however, these details just add clutter. Show return values only when they affect or alter the message flow. Or if you can't see how one object could possibly collaborate with another, show where it was returned. Omit details of how objects do low-level tasks. Stop short of explaining how preexisting objects work. Describe only how your objects use them.

Revise and rewrite. If people don't "get it," try telling them less. A designer I knew had to draw two views showing the same collaboration in order to communicate her design. One view omitted the interface details; the other included them. Some developers wanted to know what interfaces to use. Others, who only wanted to know how their parts of the system were activated, didn't want to see these details. She was delighted that by telling less (to the second group) she was able to communicate more.

The best way to "see" isn't always with a diagram. Consider complex algorithms. It's hard enough to figure out that sorting is going on by reading a sequence diagram, let alone discriminate the key aspects of a sort algorithm. A sequence of messages doesn't illustrate any side effects. You can't see what happens when an object is added to a hash table or when a buffer overflows. Diagrams can't communicate everything. You can always use words, pseudo-code, code, Backus-Naur Form grammar, decision tables, state tables, or storyboards to emphasize certain aspects of your design.

Do not overstate. Don't tell more than what you believe at any given point. For example, don't dress up a collaboration story with speculation. If you have only worked out general paths of collaboration, don't show specific messages. If you know messages, but not arguments, leave them out. Be as specific as you can, but don't state more than you will feel comfortable defending in a review. Use UML when you want to be precise, and rough sketches when you want to convey the gist of your design.

Extra precision can illuminate, but it can constrain, too. For example, on a collaboration diagram, a straight line, called a link, establishes a relationship between two collaborators. Are two linked objects both sending messages to each other? Probably the collaboration is only one-way. To make this perfectly clear, you can put a visibility arrow at the end of the link pointing to the object whose services are called upon. But if you add visibility arrows to some links, people expect them everywhere. What if you want to leave that decision open? If you add extra precision in some places and not in others, people will draw their own conclusions.

As a general principle, don't add a detail unless it adds value to your story. If you can get away with omitting spurious details, do so. For example, if you are describing the public interfaces to classes, you need not describe their attributes. Their inclusion in any diagram is always optional. Nothing requires that you include every detail. Your goal is to communicate the important aspects of your design. The more precision you add, the more difficulty you will have keeping your design stories accurate (and surprisingly, the more open they will be to misinterpretation).

Be clear. If you choose the right form of expression, your design will be more understandable. My colleague Alan McKean is great at creating "big picture" diagrams that are readily understood by almost everyone. For instance, contrast Figure 2 (on page 54), which characterizes the points of flexibility in an application that enables a severely disabled person to communicate, with the UML diagram in Figure 3 illustrating the same points. Figure 3, while explicitly calling out classes and flexible interfaces, only adds value to those interested in object design details.

For example, you can make message ordering more clear by drawing a UML sequence diagram instead of a collaboration diagram. You can annotate it to show timing, branching, looping, return values, and many other things—if these things bring clarity to your design. If they cause confusion, perhaps you need to add a running commentary.

To improve legibility, limit the number of objects and messages on a diagram to twenty-five or fewer messages between limited numbers of participants (ten objects or fewer). Or, using a collaboration diagram, you can organize objects ac-

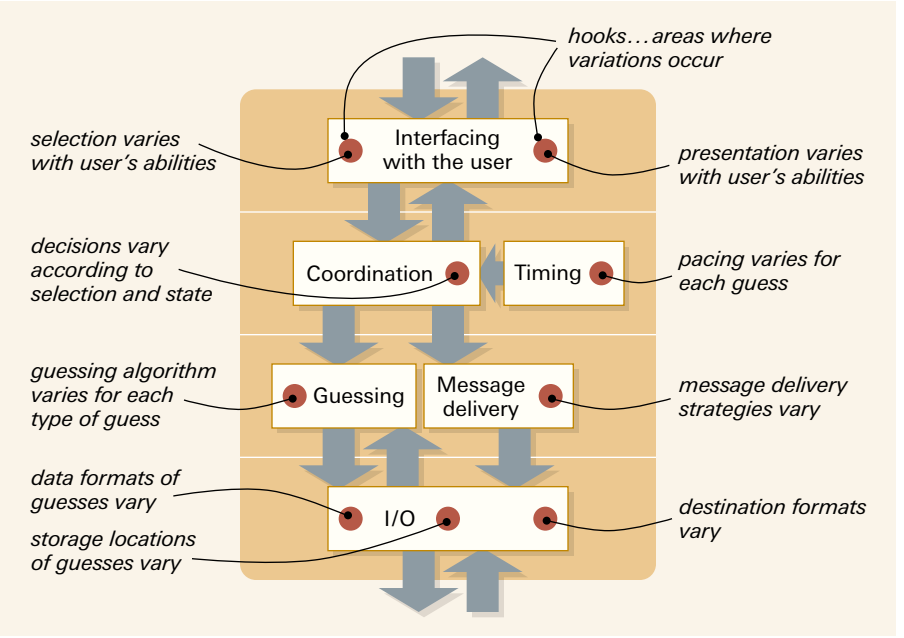

Figure 2: Use big-picture diagrams to convey information that is easy to understand.

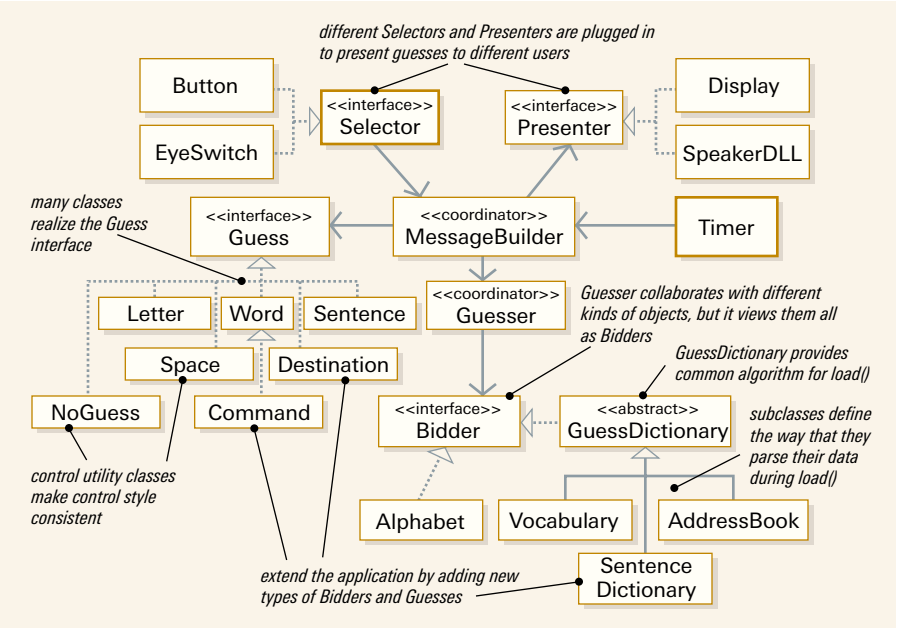

Figure 3: Choose complex diagrams such as this one for those interested in the design details.

cording to their positions in a layered architecture. This lets you see that messages follow a layered communication pattern: flowing either between objects in a given layer or from an object in a given layer to objects in adjacent layers.

Make sure the reader knows who is speaking. Don't change your point of view or add new voices to your discussion. If you are explaining how subsystems collaborate, don't drop down two levels and show objects inside those subsystems collaborating with objects from a standard library.

Parenthetical comments and notes are often spoken with a different voice and tone. When you point out things of interest too often (note: please ignore this comment), it breaks concentration. Too many parenthetical comments, cautionary notes in text, or notes on UML diagrams convince your readers that you speak hesitantly. Use these devices only when you really have something important to say and you want it to stand out.

Note: This is really, really important. Keep notes to 2% or less of what you are saying, unless you like writing stuff that nobody reads.

Do not affect a breezy manner. It isn't appropriate to leave things understated, undrawn, or unexplained. CRC cards are too breezy if you want to explain an interaction sequence. Don't arbitrarily limit your diagrams. Stick with your story. If a diagram becomes too complex, you can break it into smaller subdiagrams. UML lets you draw a dangling message arrow on one diagram (meaning the details aren't shown there) that can lead to a hanging message arrow in another diagram.

Conclusions

Although your goal is probably not to become a talented writer or visual artist, as a software developer, manager, or tester, you can apply Strunk and White's writing advice to telling your stories. An accomplished communicator uses diagrams as well as word choices to set the tone. You can get your ideas across more effectively and compactly by developing a storyline and presenting it using emphasis and progressive realization techniques. And even though you may be constrained by preexisting templates and existing documentation standards, you now know how to cut to the essence of your story. Say what you want to say, decide what deserves special emphasis, include the right details, and know when to stop. STQE

Rebecca Wirfs-Brock (rebecca@wirfs-brock.com) has pioneered many well-known object-oriented design techniques including responsibility-driven design, object role stereotypes, and the two-column form of use cases. This article is based on the chapter "Describing Collaborations" in the new book that she co-authored with Alan McKean, Object Design: Roles, Responsibilities, and Collaborations, *published by Addison-Wesley in 2003.*

***STQE* magazine is produced by Software Quality Engineering.**