The Art of Telling Your Design Story

July 2006
Rebecca J. Wirfs-Brock
Wirfs-Brock Associates
rebecca@wirfs-brock.com
www.wirfs-brock.com

The Design Process

Design is a messy, iterative process
Early descriptions tend to be less precise
Later descriptions add more precision and formality
A Design Storytelling Strategy

Tell interesting stories

Establish scope, depth and tone

Decide how detailed the story should be

Choose appropriate forms—tell it, draw it, describe it

Emphasize what’s important

Clarify, compress, and expand as needed

Scope, Depth, and Tone

Scope—how much territory you cover

Depth—how many details you present

Tone—formal, casual, educational, inspirational…
List Key Story Points

List what you want to cover—whether big, small or overlapping—and what you don’t

Key Points for “Build a Message” Collaboration
- Use a sequence diagram—keep simple
- Note complicated guessing logic
- Message content is structured
- Don’t show threading details (yet)
- Point out parts that are adaptable
A Storytelling Example

**Goal:** Describe a CRC modeling session

**Scope:** Initial design for a key use case

**Depth:** High-level paths of interactions between key objects

**Tone:** Rough, brief

- Results: Collaboration sketch with message names.
- Lists of issues and ideas

---

Early Collaboration Model Stories

Concentrates on control, coordination, important services, and information

Focus on your inventions

Stops explanation when you can demonstrate that a focused set of design elements fulfill their purpose
CRC Cards

**Candidate, Responsibilities, Collaborators**

CRC cards are an informal way to record early design ideas about candidates.

**Message Builder**
- Builds message from selections
- Presents guesses to user
- Controls the pacing

**Message Builder**

Purpose: The Message Builder is a hub of activity in the application. It coordinates the timing, the presentation of guesses, the message construction. It centralizes control and is a core element of the control architecture.

Start with rough sketches...

---

Copyright 2006, Wirfs-Brock Associates, Inc.
...then get more precise

Show a sequence of messages between these objects
- Label message arrows with names of requests
- Can be “less” than a full signature
- Shows arguments and return values when it is important to understand key information flow
A Storytelling Example

**Goal:** Explain how a design evolved

**Scope:** Control design for a key use case

**Depth:** High-level paths of interactions between key objects

**Tone:** Explanatory, moderate amount of detail

Results: Sketches of design, analysis of problem, and recommended solution
“Build A Message” Use Case

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Click</em> to start software speaking</td>
<td>Start building a message</td>
</tr>
<tr>
<td>Repeat until …</td>
<td></td>
</tr>
<tr>
<td>Optionally, “click” to select letter</td>
<td>Determine what to speak (letter, word, sentence, or space)</td>
</tr>
<tr>
<td></td>
<td>Speak letter</td>
</tr>
<tr>
<td></td>
<td>Add letter to word</td>
</tr>
<tr>
<td>Optionally, “click” to select word</td>
<td>Speak space</td>
</tr>
<tr>
<td></td>
<td>Add word to end of sentence</td>
</tr>
<tr>
<td>Optionally, “click” to select sentence</td>
<td>Start new word</td>
</tr>
<tr>
<td></td>
<td>Speak sentence</td>
</tr>
<tr>
<td></td>
<td>Add sentence to end of message</td>
</tr>
<tr>
<td></td>
<td>Start new sentence</td>
</tr>
<tr>
<td>... a command is issued</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process command (separate use cases)</td>
</tr>
</tbody>
</table>

Speak for Me enables a severely disabled user to communicate.

Build a Message

Diagram showing the interactions between Presenter, Selector, MessageBuilder, Alphabet, Vocabulary, Word, SentenceDictionary, Message, UserProfile, and Timer.
Controller = Coordinator + State

Applying the State Pattern to Simplify MessageBuilder

Each state object takes responsibility for handling events delegated to it by the MessageBuilder.

- Idling
  - Respond to user action
  - Respond to timeout
  - Know next state

- Suspended
- End of Word
- Execute Command
- Guessing Letters, Words, and Sentences
- Guessing Letters, Words, and Space
- Guessing Letters Only
Delegating Control

Factors decision-making into helper objects

Replaces complex control with simpler coordination and delegation

Distributes focused logic into classes that implement singular, smaller roles

More classes and objects
The Guess: now a more intelligent information holder

In our initial design Letters, Words and Sentences simply held the item that the user could select and knew their spoken representation

If each of these objects were responsible for adding themselves to a Message, type-based decisions could be eliminated!
Delegating Message Construction
to Guess

:Selector

handleSelection(Guess)

:Message Builder

addTo(Message)

/Guess :Letter

addLetter(Letter)

:Message

Letters, Words, Sentences, and commands can all be guesses

Message is responsible for handling specific Guesses by name

The Double Dispatch Pattern

Problem: How to select an action based on the type of two objects appearing in combination

Context: Sometimes you need to make a decision based on the class of one of the parameters in a message.

Forces: Cases or switch statements are often used in procedural languages to decide what action to take. If you hardwire in class references, you have to update your code that decides each time a new class is added. If you use polymorphism, you can eliminate decisions.

Solution: Instead of checking on a class of a parameter, add new methods having the same name to each class of a parameter objects. Pass along a reference to the calling object, then have the receiver call back and execute the “right” method.

Consequences: This eliminates any conditional checks. Code is more maintainable because it is easier to extend by adding methods rather than modifying existing ones.
To Delegating the Responsibilities

Letters, Words, and Sentences know how to add themselves to a Message.

Guesser is responsible for accessing dictionaries and making guesses.

MessageBuilder coordinates activities.

Delegating Message Construction to Guess—our preferred design

SC- Add a Guess to a Message

Presenter Selector Timer MessageBuilder Guesser Guess Letter Letter Message

handleTimeout()

presentGuess(Guess)

handleSelection(Guess)

nextGuess(Message)

Letter words, sentences, and commands are all kinds of guesses

Message is responsible for handling each different kind of guess

addLetterLetter

/before role name : before class name
Use Multiple Descriptions

- Text to explain interfaces, assumptions, qualities, ..
- Element roles and responsibilities for a conceptual overview
- Specific connections and protocol details
- UML sequence diagrams to show key interaction sequences

Use Progressive Realization

Landscape architects create views that move someone to where they want in gradual, interesting steps

With describing a high-level view of your design
  - Stick to the main points.
  - Present central objects and what is important
  - Then tell more…
Increasing Emphasis

Things gain prominence by their position and appearance. To increase an item’s emphasis:

- Put it first
- Surround it with space
- Put it in a bulleted list
- Mention it in multiple places
- Give it more room
- Repeat or restate it in different forms

Central location, size, and boldness add emphasis

Placing a controller in the middle of a collaboration draws attention to it
Speaking to A Crowd

Interests and backgrounds differ
  – Some may already know a lot
  – Some may be looking for specific facts
  – Some may want the punchline
  – Others may need fundamentals first

Choose what to emphasize. Cater to the majority; give others pointers along the way

What’s More Fundamental

Things you cannot change
Problem descriptions and requirements are more fundamental than solutions

The typical case
Things are more fundamental than relations between them
Dealing with an Impatient Audience

Those seeking specific facts won’t sit still:
- Include a Frequently Asked Questions section
- Don’t always present fundamentals first
- Present important things that deserve emphasis

Hold their Attention

Descriptions are more clearly understood if important parts are told first

Be aware of monotony setting in. Counteract it:
- Explicitly call out details
- Point out that the next five diagrams are similar
- Illustrate different perspectives
Describing Variations

What does this mean?
– (a) road slippery when wet
– (b) dangerous curve
– (c) reduce speed
– (d) drunk driver ahead
Describe Only What The Person Making the Variation Needs to Know

If the support you have provided is high, then someone making a variation may not require deep knowledge.
– Perhaps you created a number of pre-built components or classes. To implement a variation, a developer chooses an existing component and plugs it into a collaboration by invoking a single “setter” method.
– They only need a simple set of instructions.

A Basic Recipe for Installing a New Device Driver:

Choose from one of ....
Configure into system by doing ....
Restart and test by calling........

Showing How to Adapt a Design

Explain three things:
– the current design
– what aspects are adaptable
– how to make these adaptations
What Can Be Configured

What Can Be Guessed

Copyright 2006, Wirfs-Brock Associates, Inc.

The Art of Telling Your Design Story
Recipe for Adding a Guess

To Add a New Kind of Guess

1. Define a class that implements the Guess interface.
   This type of object must know contents, formatted for both display and speech, know how long to wait before continuing with another guess, and be able to add itself to a message.
   Specifically, it must implement these methods:
   ```java
   public String displayableText()
   public String speakableText()
   public String getContent()
   public Duration waitTime()
   void processMessage(Message m)
   ```

2. Define a class that implements the Bidder interface.
   This type of object will contain all of the corresponding Guess objects and determine which is most relevant to the current message and how relevant they are. Specifically, it must implement:
   ```java
   Bid bidOn(Message m)
   ```

---

The Basic Elements of A Recipe*

**Recipe Name:** Usually starts with “How to”

**Intent**  The reason to use this recipe

**Design Description** Which classes and interfaces are involved and need to be understood, what roles do they play and what collaborations are involved. What responsibilities are adapted via the variation... backed up by supporting UML diagrams and other descriptions.

**Related Recipes:** Alternative ways to accomplish a similar variation; or related sub-recipes. A complex recipe may be broken down into several sub-recipes.

**Steps** 1. First create a class that implements the xyz interface...

   2. In it define a method named...

   3. And another method named...

**Discussion:** Mention problems that might crop up, how to test that a variation is correctly installed, or what should not be attempted using this approach.

* from *The UML Profile for Framework Architectures*

by Marcus Fontoura, Wolfgang Pree and Bernhard Rumpe
Strunk and White’s Elements of Style

Do not overwrite. 10 pictures are not worth 10k words
  – If collaborations are similar, show a typical case first, then note how remaining ones differ.
  – Draw representative interactions

Strunk and White’s Elements of Style

Omit needless words.
  – Visual equivalents of “needless words” on a collaboration or sequence diagram:
    • Gratuitous return values
    • Internal algorithmic details
    • Details of caching and lazy initialization
    • Object creation and destruction
Bubble Sort Explained

The algorithm for a bubble sort consists of two nested loops. The inner loop traverses the array, comparing adjacent entries and swapping them if appropriate, while the outer loop causes the inner loop to make repeated passes. After the first pass, the largest element is guaranteed to be at the end of the array, after the second pass, the second largest element is in position, and so on. That is why the upper bound in the inner loop decreases with each pass; we don’t have to revisit the end of the array.
Bubble Sort: A Visual Illustration

Consider the array 42,56,13,23
Let’s start sorting………

42,56,13,23  no swap
42,56,13,23  swap
42,13,56,23  swap – end of 1st pass outer loop
42,13,23,56  swap
13,42,23,56  swap – end of 2nd pass outer loop
13,23,42,56  no swap – end of 3rd pass

Bubble Sort: Some Code

```java
class BubbleSorter{
    void sort(int a[])
    {
        for (int i = a.length; --i>=0; )
        {
            boolean swapped = false;
            for (int j = 0; j<i; j++) {
                if (a[j] > a[j+1]) {
                    int T = a[j];
                    a[j] = a[j+1];
                    a[j+1] = T;
                    swapped = true;
                }
            }
            if (!swapped) return;
        }
    }
}
```
Where UML Diagrams Fall Short

The best way to see isn’t always with a standard diagram

Use words, pseudo-code, code, BNF grammar, decision tables, state tables, or pictures that emphasize certain features

Sequence diagrams sometimes fall short:
– They do not show side-effects
– It’s hard to emphasize special areas
– It’s hard to interpret algorithms
– Control flow and iteration can be difficult to see

Strunk and White’s Elements of Style

*Revise and rewrite.* If someone doesn’t “get it”, it could be your problem
– A designer drew two views showing the same collaboration. One view omitted the interface details, the other included them. Some developers wanted to know what interfaces to use. Others who only wanted to how their parts of the system were activated didn’t want to see these details. Both views were needed to get her design across.
Consider Your Audience

- selection varies with user's abilities
- decisions vary according to selection and state
- guessing algorithm varies for each type of guess
- data formats of guesses vary
- storage locations of guesses vary
- presentation varies with user's abilities
- pacing varies for each guess
- message delivery strategies vary
- destination formats vary

hooks ... areas where variations occur

The Same View In UML

- different Selectors and Presenters are plugged in to present guesses to different users
- many classes realize the Guess interface
- GuessDictionary provides common algorithm for load()
- subclasses define the way that they parse their data during load()
- extend the application by adding new types of Bidders and Guesses
- extend the utility classes make control style consistent
- Guesser collaborates with different kinds of objects, but it views them all as Bidders

Copyright 2006, Wirfs-Brock Associates, Inc. The Art of Telling Your Design Story
Strunk and White’s Elements of Style

**Do not overstate.** Don’t tell more than what you believe at any given point

– The same diagram can be drawn with different degrees of precision
  • If you only know general paths of collaboration—don’t show specific messages
  • If you know specific messages, but not the arguments—don’t invent arguments just to fill in the blanks

Strunk and White’s Elements of Style

**Do not affect a breezy manner.** Don’t 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 to a single page, or to ten or less objects. Stick with your design description goals. Get it down, then figure out how to show it
Strunk and White’s Elements of Style

*Be clear.* Choose the right form of expression
- To emphasize message ordering, use a sequence diagram. If timing is critical, add timing marks
- Add running commentary to explain

**A picture doesn’t tell all**

Acts as a key to supporting documentation
Can be understood at a glance
Is explicit
Provokes discussion
### Design Representation Options

<table>
<thead>
<tr>
<th>Goal</th>
<th>Simple Representation</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describe responsibilities and collaborators</td>
<td>CRC cards</td>
<td>Transfer information to a document/use a table</td>
</tr>
<tr>
<td>Show collaboration relationships between objects</td>
<td>Simple Collaboration Diagram</td>
<td>Add visibility links to make explicit who collaborates with whom</td>
</tr>
<tr>
<td>Illustrate an Interaction Sequence</td>
<td>Collaboration Diagram</td>
<td>To be more formal, draw a sequence diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To explain how objects are affected, add a running commentary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To explain interactions between subsystems treat them as “big objects” and include in diagram</td>
</tr>
</tbody>
</table>

### Design Representation Options

<table>
<thead>
<tr>
<th>Goal</th>
<th>Simple Representation</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explain complex algorithms</td>
<td>A visual animation or storyboard</td>
<td>Pseudo-code</td>
</tr>
<tr>
<td>Describe detailed interactions</td>
<td>Either a collaboration or sequence diagram</td>
<td>Draw an interaction diagram and annotate</td>
</tr>
<tr>
<td>Describe how to reconfigure a collaboration</td>
<td>Define responsibilities of “configurable objects”</td>
<td>Write a recipe describing a step-by-step procedure for configuring a collaboration</td>
</tr>
<tr>
<td></td>
<td>Draw a typical interaction sequence. Identify where configurable alternates can be “plugged” in</td>
<td>Include examples or sample code</td>
</tr>
</tbody>
</table>

Copyright 2006, Wirfs-Brock Associates, Inc.
Read more about describing collaborations in:

*Object Design: Roles, Responsibilities and Collaborations*, Rebecca Wirfs-Brock and Alan McKean, Addison-Wesley, 2003

www.wirfs-brock.com for articles & resources & blog