Skills for the Agile Designer

Rebecca Wirfs-Brock
Wirfs-Brock Associates
rebecca@wirfs-brock.com

Agenda

- What is agility?
- Tools for agile designers
  - Seeing
  - Shaping
  - Describing
- Agility and Design Rhythms

Agility Recognizes Design Values

- Agility doesn't demand that you throw away what works. Instead, it goes hand-in-hand with articulating your design values. Agile designers acknowledge that:
  - Modeling should have a specific purpose
  - Documentation should add value
  - Reflection is crucial
    - You can't get better if you don't measure the effects of changing your work products, design process, or development practicesReplanning is essential
    - Agility isn't a single "event"; it's a process

Disturbing Beliefs and Trends

- Agility does not mean that:
  - Problem solving must occur in two week increments
  - You only talk informally about code "smells" instead of having in-depth design discussions and review
  - You equate refactoring code with good design (and don't design before you code)
  - You do whatever works without regard to any process. Formal = bad; Informal = good
  - Design descriptions and documentation add no value

What Fueled these Trends

- Analysis paralysis
- Endless high-level discussions with little regard for implementation concerns
- Code restructuring *does* make things better. Design doesn't stop when coding starts and cleaning up code is part of preserving design integrity
- The hope that if we omit extra stuff that adds no value maybe, just maybe, we'll succeed at getting to market sooner and satisfying our customers. Being formal or precise takes time.
- Piles of models. No useful results. Many design artifacts don't add value, so why not cut them out?

Agility Demands Design Integrity

- Agile designers employ strategies for designing responsibly in the face of uncertainty, complex problems with no obvious answers, and lots of tedious details
- Agile designers (and their managers) need to react and respond to changing situations, while steadily working on all aspects of their design
- Cutting corners isn't the goal, cutting out inefficiency is the goal
- A process that works on one project can't be blindly applied to the next project. Agility is not a recipe. It's a journey.

Tools for Seeing

A Thought Experiment

What if we believed that only those endowed with an innate gift could learn to read? What if teachers believed the best way to instruct was to expose a child to lots of reading materials, then wait to see what happens? Fear of stifling creativity would dampen attempts to guide new readers. If a child asked how to read something, a teacher might say, 'Try whatever you think works. Enjoy it, explore, reading is fun!' Perhaps one or two in any class would possess that rare talent and spontaneously learn to read.

8

How Do You Learn to Draw?

- "It is simply a matter of learning basic perceptual skills—the special ways of seeing required for drawing. I claim that anyone can learn enough *seeing* skills to draw a good likeness of something seen 'out there' in the real world."

  --Betty Edwards

Overcoming our Biases

"Perhaps artistic [drawing] talent has seemed rare and out of the ordinary because we expect it to be rare and out of the ordinary."

--Betty Edwards, *Drawing on the Artist Within*

- The same claim might be made for:
  - Thinking abstractly
  - Developing a well-factored object design based on object roles
  - Creating application-specific frameworks

How *Do* You Learn to Design?

- You become adept at object design with enough practice and experience seeing the nature of the design problem and learning fundamental strategies for producing an acceptable solution.
- Although a highly creative activity, design fundamentals can be easily learned

Fundamental Design Constructs

- an application = a set of interacting objects
- an object = an implementation of one or more roles
- a role = a set of related responsibilities
- a responsibility = an obligation to perform a task or know information
- a collaboration = an interaction of objects and/or roles
- a contract = an agreement outlining the terms of a collaboration

Object Role Stereotypes

- "Purposeful oversimplifications" help us focus an object's responsibilities
  - Information holder - knows and provides information
  - Structurer - maintains relationships between objects and information about those relationships
  - Service provider - performs work and, in general, offers services
  - Coordinator - reacts to events by delegating tasks to others
  - Controller - makes decisions and closely directs others' actions
  - Interfacer - transforms information and requests between distinct parts of a system

Seeing leads to questioning...

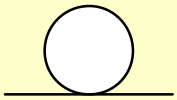
- "I'm curious about something. Though I noticed it somewhat before, it seems very prominent in our midterm example. I think my question is general enough not to worry about broadcasting it to the group.

  Are domain objects often primarily information holders? It seems most domain objects are things (Customers, Participants, Events, etc) and things tend to know information about themselves. Or also service providers sometimes. I don't see many Coordinators or Controllers as domain objects. Those seem to usually be behind the scenes (I.e. system level) objects.

  Again, I'm just curious if this is common, or if it's a mistake I'm making when I think about what a 'domain' object should be."

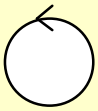
Or does it?

Early analysis **object** stereotypes from Objectory……



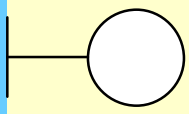
<<entity>>

An entity is passive. It does not initiate any action.



<<control>>

A control object controls interactions between a collection of objects. It usually corresponds to a use case.

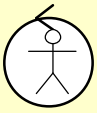


<<boundary>>

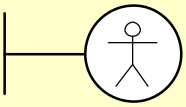
A boundary object lies on the periphery of the system. It interacts with all 3 kinds of objects.

The Power of Blends

Stereotypes from Objectory business modeling..

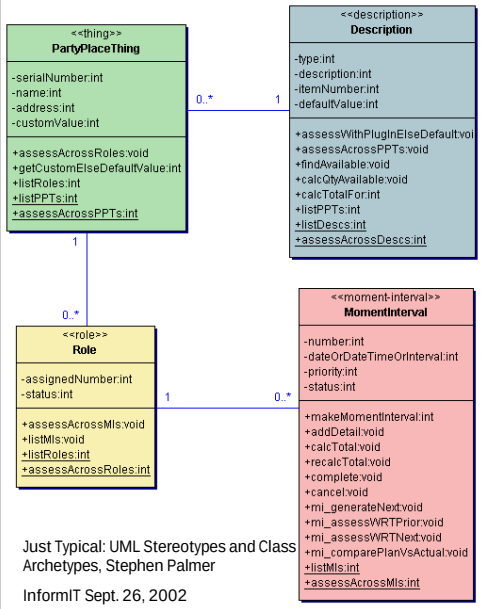


A worker represents an abstraction of a human that acts within the system.



A case worker is a worker that interacts directly with others outside the system.

How Peter Coad Sees…



Just Typical: UML Stereotypes and Class Archetypes, Stephen Palmer

InformIT Sept. 26, 2002

Pink <<moment-interval>> represents events or activities that we need to keep track of for business or legal reasons

Yellow <<role>> classes represent a way of participating in an event or activity by a green entity

Green entity classes are categorized into <<party>>, <<place>>,and <<thing>>

Blue <<description>> represents a description of some physical thing

17

Seeing Should Generate Ideas

- Classification is useful… but generating ideas about behavior is productive
  - "Black-and-white conveys basic information. Color reaches out and grabs you."–Peter Coad
  - "…archetype[s] have more or less the same sort of attributes and operations and tend to interact with other archetypes in generally predictable ways. These patterns of characteristics and behavior can help us very … quickly identify attributes and operations… and give us increased confidence in the structure of our code."–Stephen Palmer

18

The Power of Seeing in Color

- "When working informally, I use four pads of different colored, three-inch square Post-it notes to represent classes on flipchart pads drawing associations with a suitable marker pen.

  In the same way that after you have watched color television or seen color photographs, you do not want to return to black and white except for nostalgic reasons, most people never want to return to the shallowness of the black-and-white version once they have experienced object modeling in color."

  --Stephen Palmer

The Generative Power of Role Stereotypes to Identify Responsibilities

- Pushing on an object's character leads to initial responsibilities.
  - Ask of a service provider, "what requests should it handle?" Turn around and state these as responsibilities for "doing" or "performing" specific services
  - What duties does an interfacer have for translating information and requests from one part of the system to another (and translating between different levels of abstraction)
  - What important events does a controller handle and what others does it direct... ?

Seeing at Different Abstraction Levels

- We can see "objects and behavior" at different levels. Martin Fowler* identifies 3 levels:
  - At the *conceptual* level, an object is a set of responsibilities
  - At the *specification* level, an object is a set of methods that can be invoked by other objects or itself
  - At the *implementation* level, an object is code and data

Design Patterns Explained,

Alan Shalloway & James R. Trott, pp. 15-16

Pulling Up a Level

- Reverse engineer a class into responsibilities…

The Java Calendar class

Internally, Calendar keeps track of a point in time in two ways. First, a "raw" value is maintained, which is simply a count of milliseconds since midnight, January 1, 1970 GMT, or, in other words, a Date object. Second, the calendar keeps track of a number of fields, which are the values that are specific to the Calendar type. These are values such as day of the week, day of the month, and month. The raw millisecond value can be calculated from the field values, or vice versa.

Calendar also defines a number of symbolic constants. They represent either fields or values. For example, MONTH is a field constant. It can be passed to get() and set() to retrieve and adjust the month. AUGUST, on the other hand, represents a particular month value. Calling get(Calendar.MONTH) could return Calendar.AUGUST.

Calendar Methods

public int getFirstDayOfWeek()

This method returns the day that is considered the beginning of the week for this Calendar. This value is determined by the Locale of this Calendar. For example, the first day of the week in the United States is Sunday, while in France it is Monday.

public abstract int getGreatestMinimum(int field)

This method returns the highest minimum value for the given time field, if the field has a range of minimum values. If the field does not have a range of minimum values, this method is equivalent to getMinimum().

public abstract int getLeastMaximum(int field)

This method returns the lowest maximum value for the given time field, if the field has a range of maximum values. If the field does not have a range of maximum values, this method is equivalent to getMaximum(). For example, for a GregorianCalendar, the lowest maximum value of DATE_OF_MONTH is 28.

public abstract int getMaximum(int field)

This method returns the maximum value for the given time field. For example, for a GregorianCalendar, the maximum value of DATE_OF_MONTH is 31.

public final void set(int year, int month, int date)

This method sets the values of the year, month, and day-of-the-month fields of this Calendar.

public final void set(int year, int month, int date, int hour, int minute) This method sets the values of the year, month, day-of-the-month, hour, and minute fields of this Calendar.

public final void set(int year, int month, int date, int hour, int minute, int second)

This method sets the values of the year, month, day-of-the-month, hour, minute, and second fields of this Calendar.

public void setFirstDayofWeek(int value)

This method sets the day that is considered the beginning of the week for this Calendar. This value should be determined by the Locale of this Calendar. For example, the first day of the week in the United States is Sunday; in France it's Monday.

public void setLenient(boolean lenient)

This method sets the leniency of this Calendar. A value of false specifies that the Calendar throws exceptions when questionable data is passed to it, while a value of true indicates that the Calendar makes its best guess to interpret questionable data. For example, if the Calendar is being lenient, a date such as March 135, 1997 is interpreted as the 135th day after March 1, 1997.

public void setMinimalDaysInFirstWeek(int value)

This method sets the minimum number of days in the first week of the year. For example, a value of 7 indicates the first week of the year must be a full week, while a value of 1 indicates the first week of the year can contain a single day. This value should be determined by the Locale of this Calendar.

public final void setTime(Date date)

This method sets the point in time that is represented by this

Calendar.

public void setTimeZone(TimeZone value)

This method is used to set the time zone of this Calendar.

To Get The Big Picture: Calendar Revealed

Calendar	
Represents a calendar system	Date
Performs date arithmetic	Locale
Compares dates	
Knows locale information	

Pulling Up a Level

- Examine and characterize a design… to see and understand… and then change it

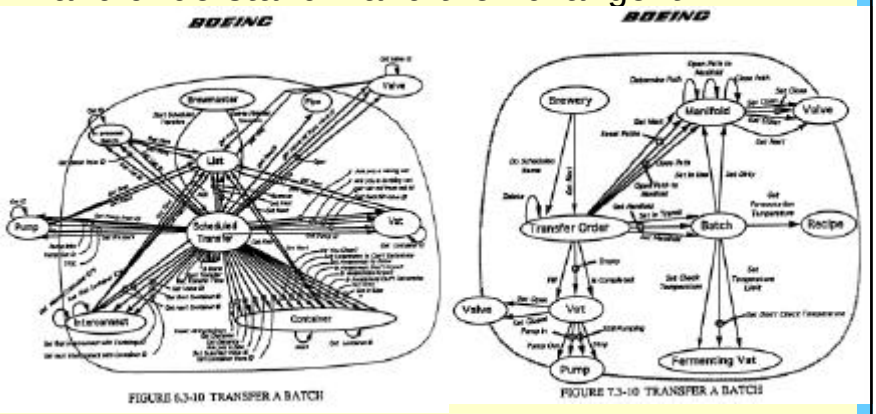


FIGURE 6.3-10 TRANSFER A BATCH

FIGURE 7.3-10 TRANSFER A BATCH

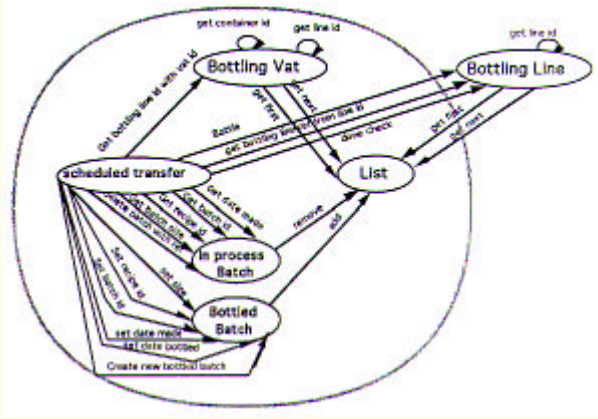
"The Object-Oriented Brewery: A Comparison of Two Object-Oriented Methods," R. Sharble and S. Cohen, Boeing Technical Report BCS-G4059, 1992.

"How Designs Differ", R. Wirfs-Brock, Smalltalk Report, vol. 1, no. 4

To Characterize the Two Designs

Data-Driven	Responsibility-Driven
centralized control	delegated control
controllers	coordinators
inherited attributes	inherited behavior
many low-level messages	fewer, higher-level messages
lots of simplistic information holders	a few smart objects that blend role stereotypes

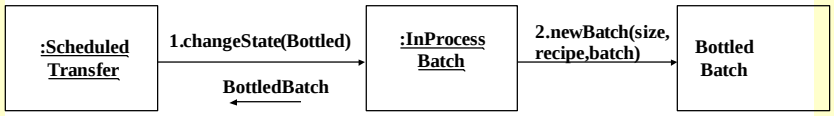
Bottling a Batch: Centralized Control



Then Work to Improve Encapsulation

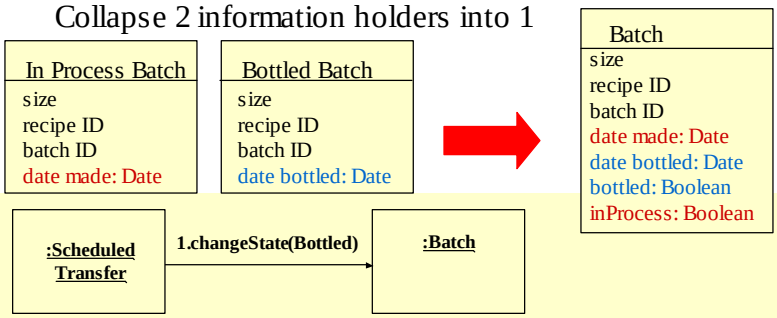
Approach #1

Delegate creation of Bottled Batch to In Process Batch



Approach #2

Collapse 2 information holders into 1



27

Tools for Shaping Solutions

Frame Your Design Problems

"When you turn on a light, you probably think of your movement of the control button and the illumination of the light as a single event. In fact, of course, something more complex is going on."

– Michael Jackson

- Software systems can be thought of a set of related and interconnected subproblems– and as a consequence may be comprised of several different "problem frames". Each different class of problem has different concerns and design issues.

5 Problem Frames

- **Control Problems** - controlling state changes of external devices or machinery.
- **Connection Problems** - receiving or transmitting information indirectly through a connection.
- **Information Display Problems** - presenting information in response to queries about things and events known by your software.
- **Workpiece Problems** - a tool that allows users to create and manipulate computer-processable objects or "workpieces". Just like a lathe is a tool for woodworking, software help users create documents, compile programs, compose music, perform calculations, manipulate visual images, generate reports...
- **Transformation Problems** - converting some input to one or more output formats.

Problem Frames And Design Focus

- Each different class of problem has different concerns and design issues
- Control frames–Do you need to determine whether attempts at changing external conditions had the desired effect? If so, you will design ways to probe whether things are as you expect. And if they aren't, well… is the problem your software or an external device, and how should you recover?
- Connection problems--Connections break down, information gets lost or gets garbled. You may need to re-establish connections or try alternate paths, or…

Problem Frames and Design Focus

- Information Display Problems--Does your design have to accommodate imprecise questions or partial answers?
- Workpiece Problems–What's the real nature of the workpiece and how usable is your tool? Do we really need all the features of Microsoft Word to edit documents?
- Transformation Problems--What constitutes an acceptable loss of information or the reversibility of a transformation can be an issue. What constraints are there on speed or memory utilization?

Problem Frames and Design Focus

- The ideal: Jackson advocates fully understanding the nature of the problems your software is trying to solve before you start design .
- The agile reality: In a world full of imperfect knowledge and time constraints, be better prepared by characterizing what problems your design must solve… more problems may crop up during design.

How a Connection Problem Affects Your Design

"In many problems you'll find that you can't connect the [software] machine to the relevant parts of the real world in quite the way you would like. You would prefer a direct connection...instead you have to put up with an indirect connection that introduces various kinds of delays and distortion."
–Michael Jackson

- Basic strategies for dealing with connection issues:
  - Consider that your software is really interacting with "something in the middle" that is connected to "something out there" that doesn't always work
  - Design your software to react in the face of potential time-delays, conflicting states between "connected" system as well as faulty connections.

Agility and Collaboration Design

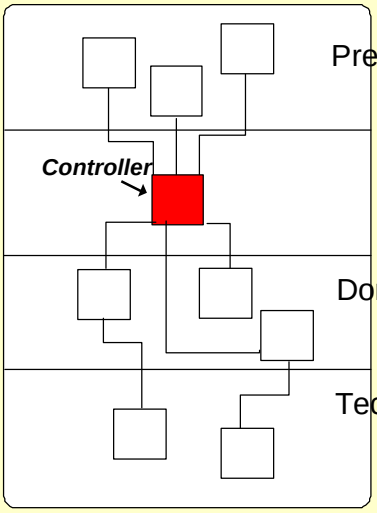
- Traits of an agile designer
  - Doesn't fudge on design complexity
  - Doesn't blindly apply design patterns as "the best solution" to a problem
  - Explores alternatives
  - Looks for simplifying solutions
  - Looks to develop a consistent style
  - Has a sense of aesthetics (and compromises them when time gets tight)
  - Uses patterns as archetypes

"Build A Message" Use Case

Speak for Me enables a severely disabled user to communicate

Actor Actions	System Responsibilities
"Click" to start software speaking	Start building a message
Repeat until . . .	
Optionally, "click" to select letter	Determine what to speak (letter, word, sentence, or space) Speak letter Add letter to word
Optionally, "click" to select word	Speak space Add word to end of sentence Start new word
Optionally, "click" to select sentence	Speak sentence Add sentence to end of message Start new sentence
… a command is issued	
	Process command (separate use cases)

Centralized Use Case Control



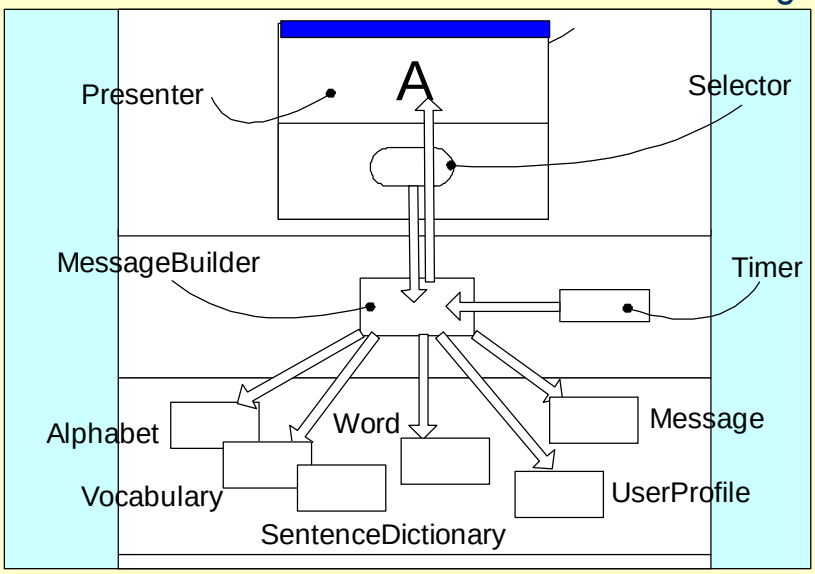
Presentation Layer

Application Layer

Domain Services Layer

Technical Services Layer

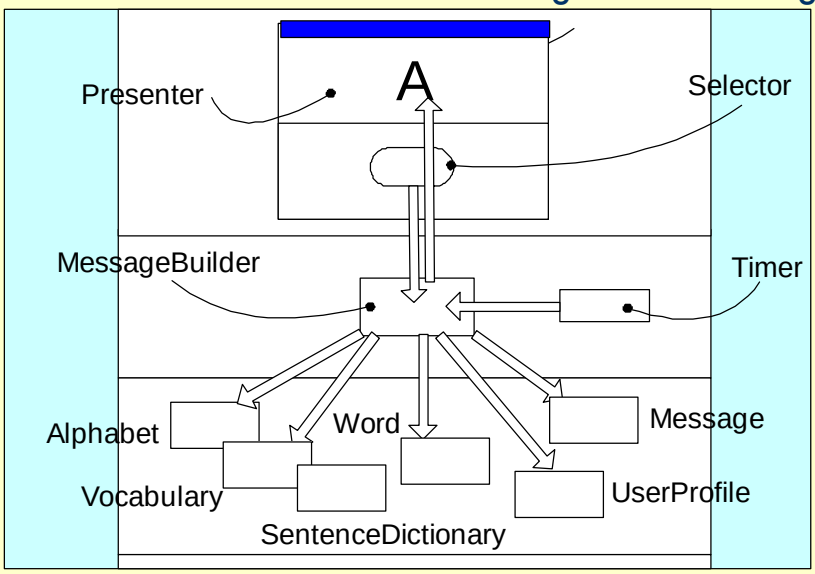
Build a Message



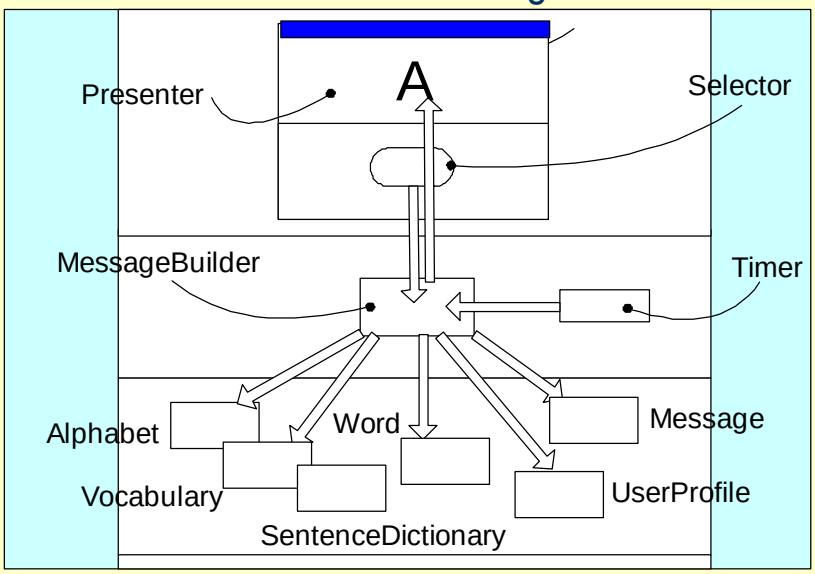
Centralizing Control

- Many decisions make the controller difficult to manage …
- What does it do when a user selects something? It could be a letter, a word, a space, a sentence, a command, a destination
- When does it present each of the above to the user? It depends on the state of the message, what the user did last, and on the state of the software

Controlling the Guessing



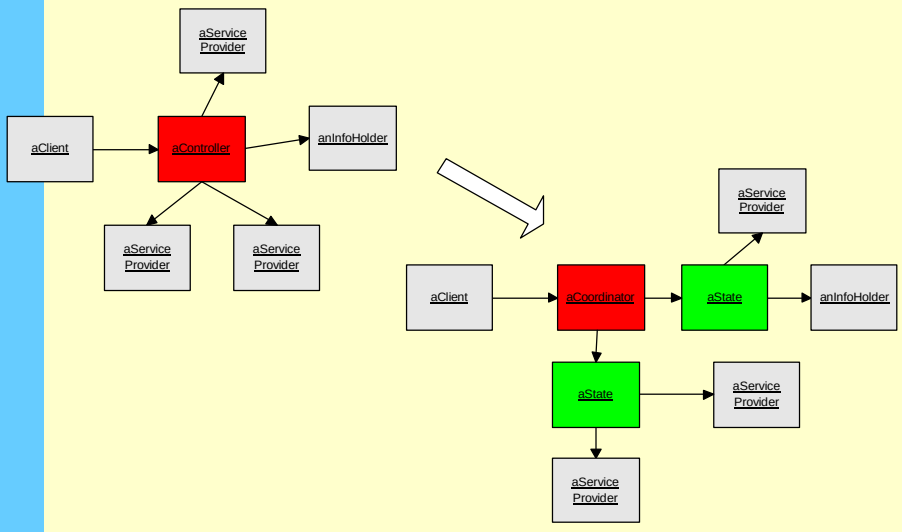
Handling the Selections



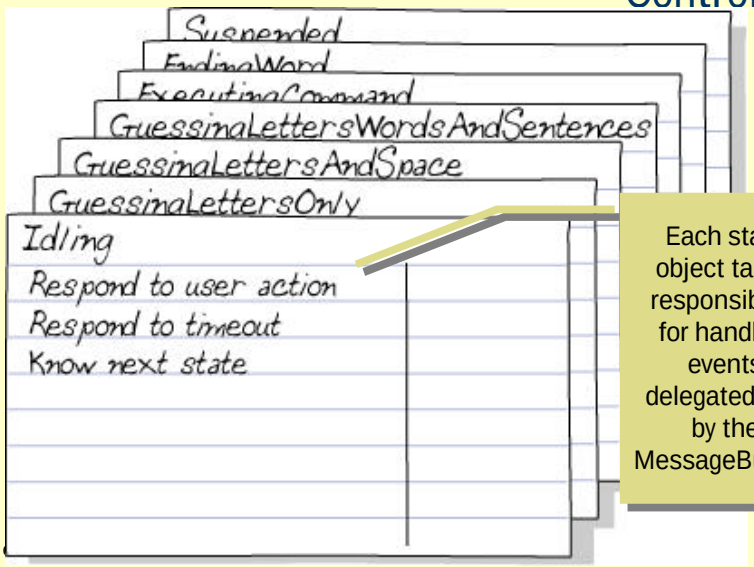
Delegating Control

- 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

Controller = Coordinator + State



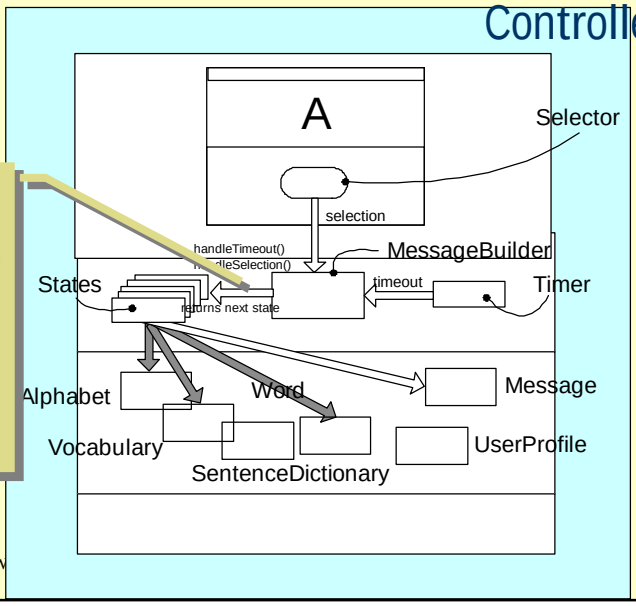
Applying the State Pattern to Simplify the Controller



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

Applying the State Pattern Simplifies the Controller

Complexity is still located in a "control center". Can we simplify the control center by giving other objects more responsibility?

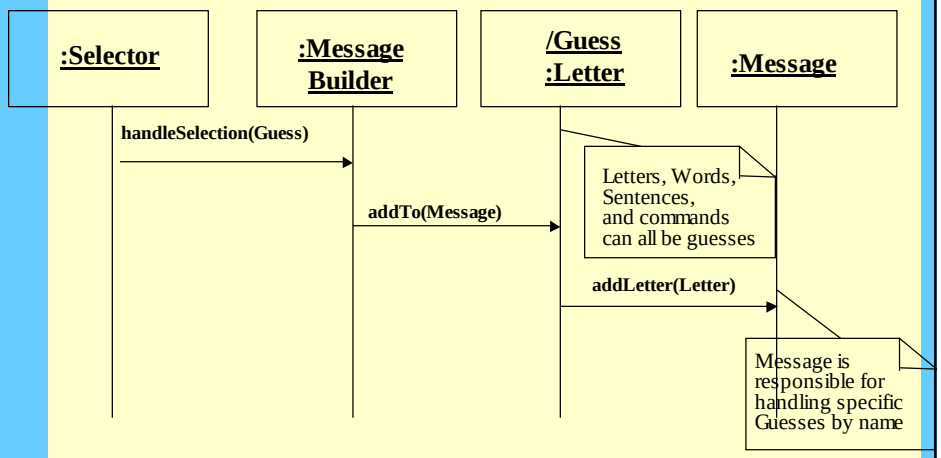


The Guess

now a more intelligent information holder

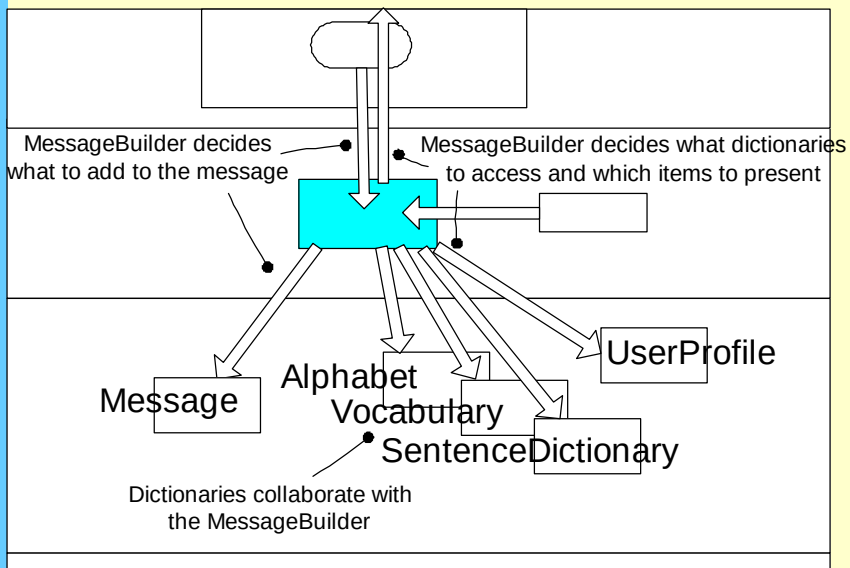
- In our initial thinking, Letters, Words and Sentences didn't have many responsibilities. They simply held the item that the user could select and knew their spoken representation.
- But if Guesses directly add themselves to a Message, then extra messaging can be eliminated! We prefer this design choice because it reduces complexity.
- It doesn't eliminate the need for Message to know about different kinds of Guesses, but it simplifies patterns of collaboration.

Delegating Message Construction to Guess– our preferred design



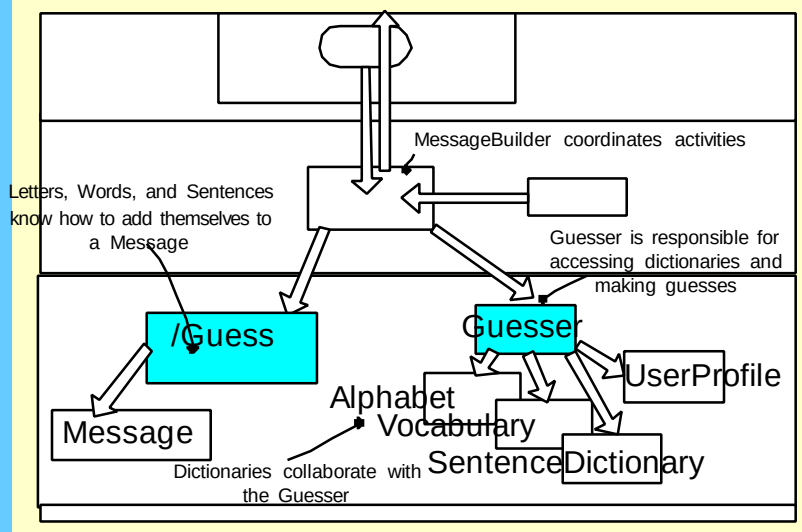
47

From Controlling Everything ...



48

To Delegating the Responsibilities



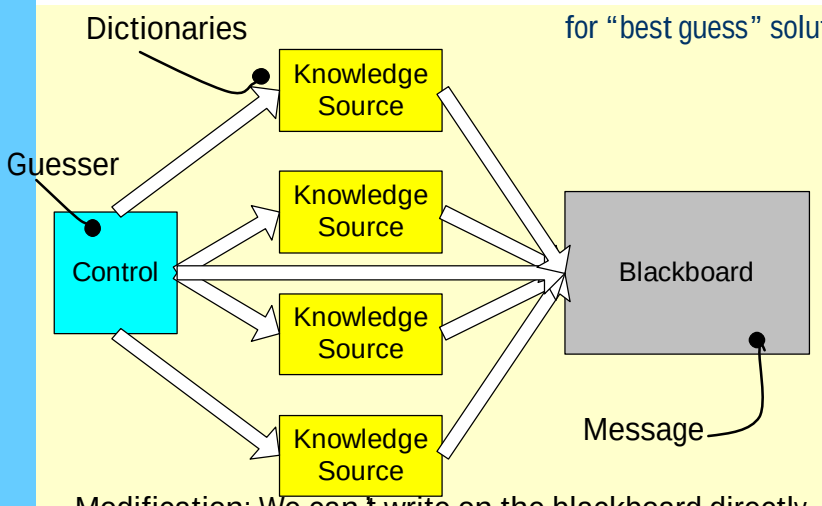
Delegation to The Guesser

a new invention

- A Guesser object assumes all of the responsibilities for guessing that were previously performed by the controller (MessageBuilder)
- It hides all of the mechanisms of guessing, providing a black box for developing the guessing machinery
- Since guessing is a complex task, maybe the Guesser can delegate some of its responsibilities, too?

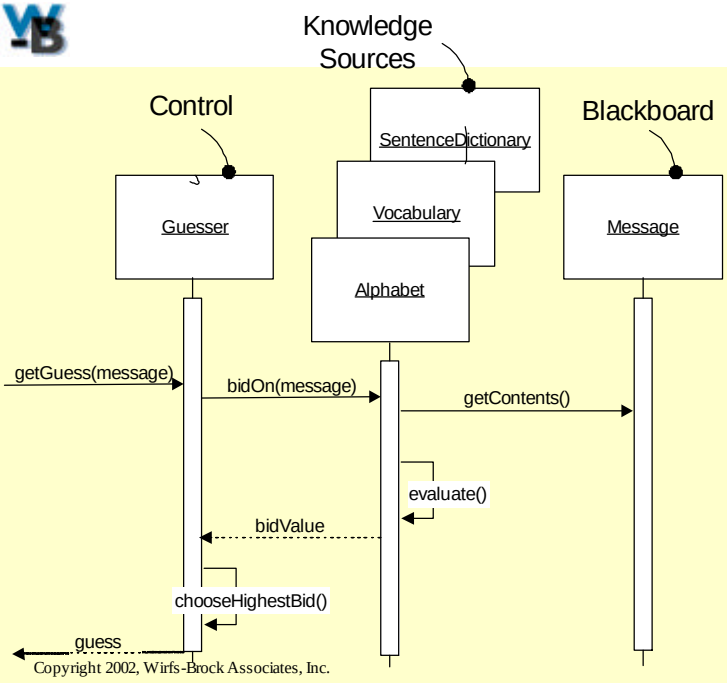
The Blackboard Architecture

for "best guess" solutions



Modification: We can't write on the blackboard directly. Only after the user chooses a "guess" can we add it to the message.

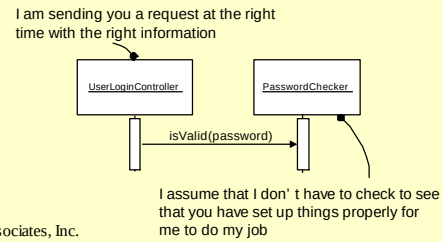
51



52

Trust Regions

- Carve your software into regions where "trusted communications" occur
- Objects in the same trust region communicate collegially, although they may still encounter exceptions and errors



Collaboration Cases To Consider

- Collaborations between objects...
  - that interface to the user and the rest of the system
  - inside your software and objects that interface to external systems
  - in different layers or subsystems
  - you design and objects designed by someone else

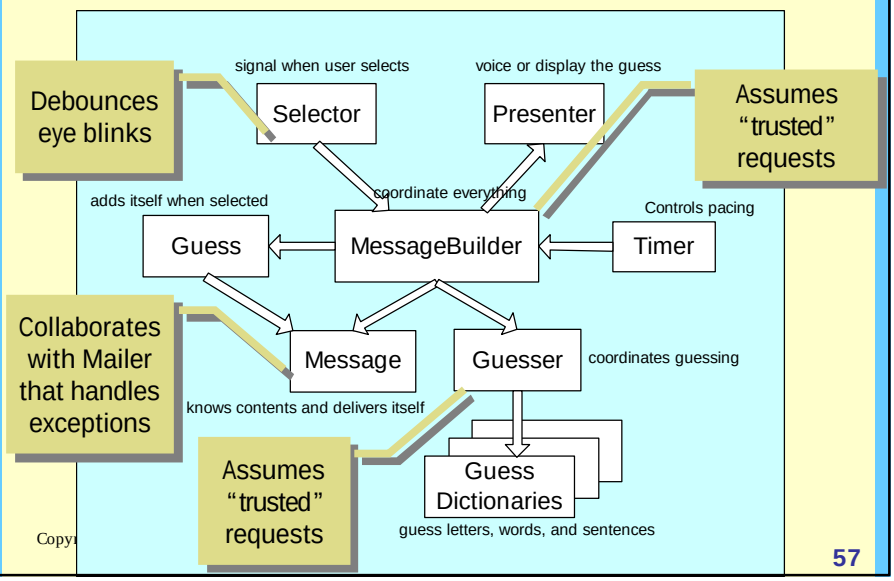
Using An Untrusted Collaborator

- Extra precautions may need to be taken. Especially if the client is responsible for making collaborations more reliable
  - Pass along a copy instead of sharing data
  - Check on conditions after the request completes
  - Employ alternate strategies when a request fails

Implications of Trust

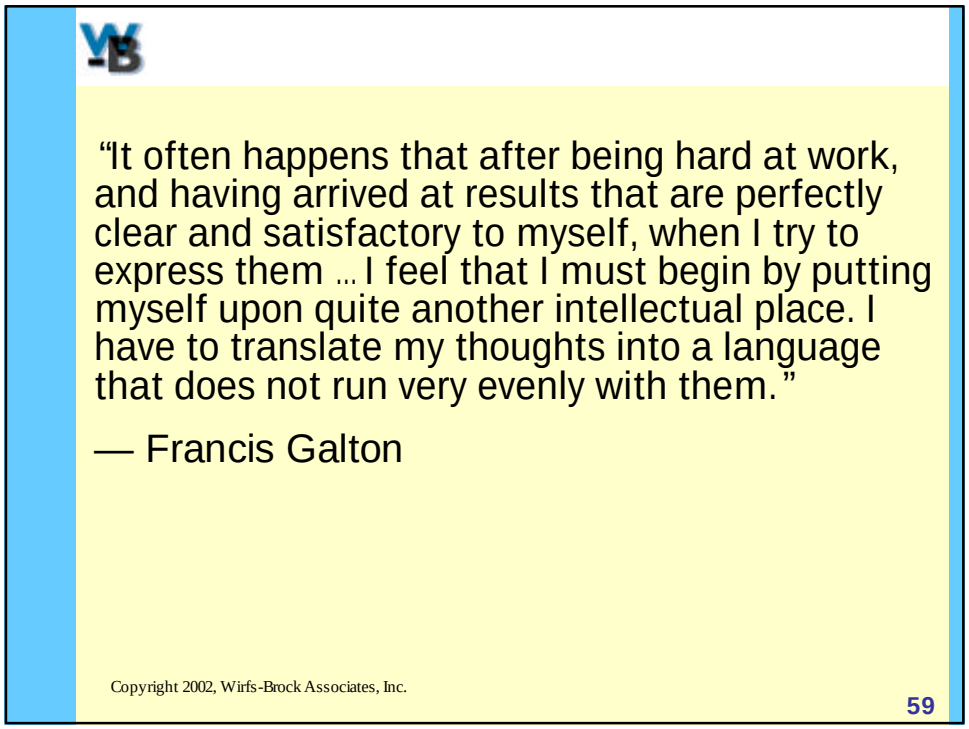
- In Speak for Me, all objects in the application "core" are within the same trust region
- Objects in the application control and domain layers assume trusted communications between each other
- Objects at the "edges"—within the user interface and in the technical services layers—make sure outgoing requests are honored and incoming requests are valid

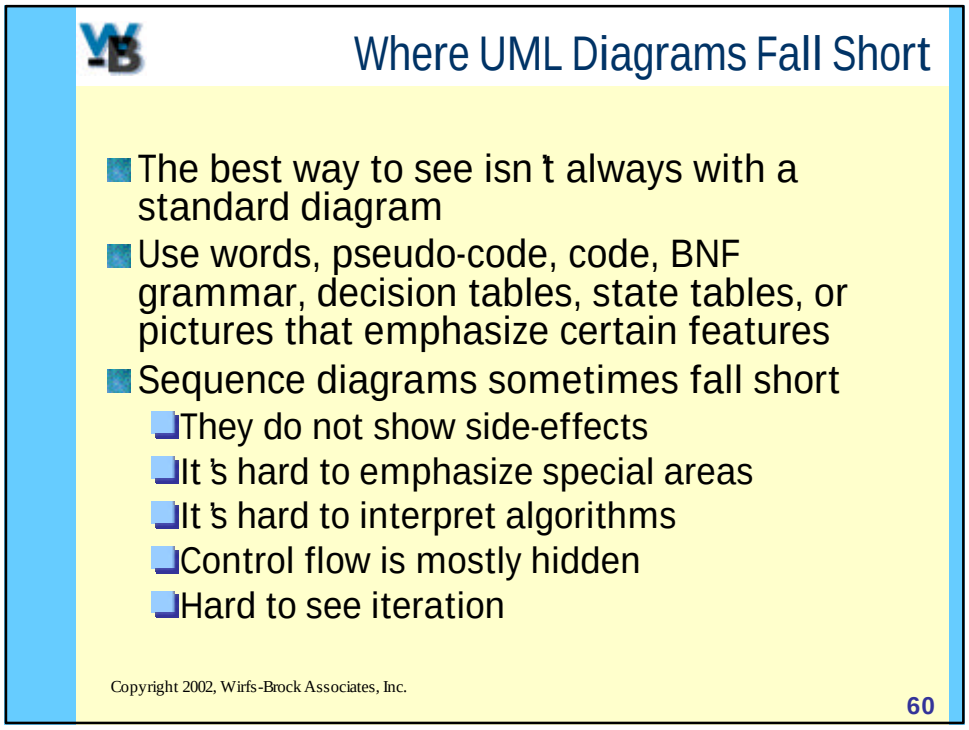
Objects At The "Edges" Take On Added Responsibilities



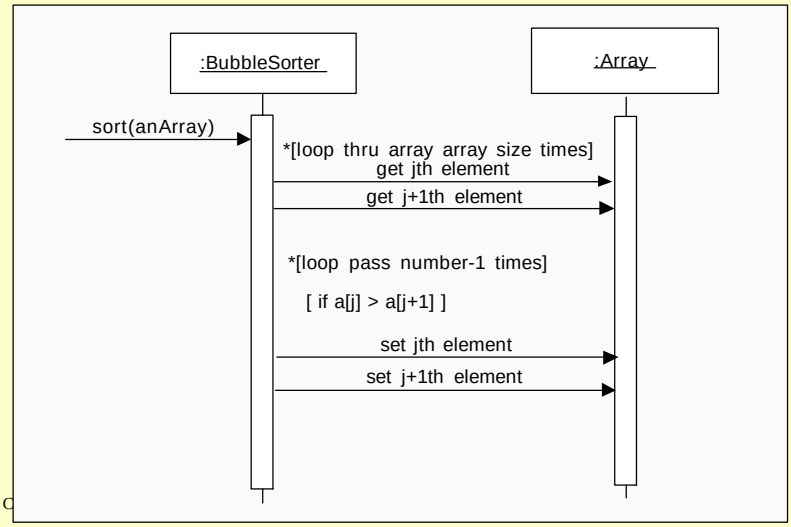
57

Tools for Describing Designs


"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 place. I have to translate my thoughts into a language that does not run very evenly with them."
— Francis Galton
Copyright 2002, Wirfs-Brock Associates, Inc.
59


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 is mostly hidden
Hard to see iteration
Copyright 2002, Wirfs-Brock Associates, Inc.
60

Bubble Sort:
A UML Sequence Diagram



61

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.

62

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

```
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; } } }
```

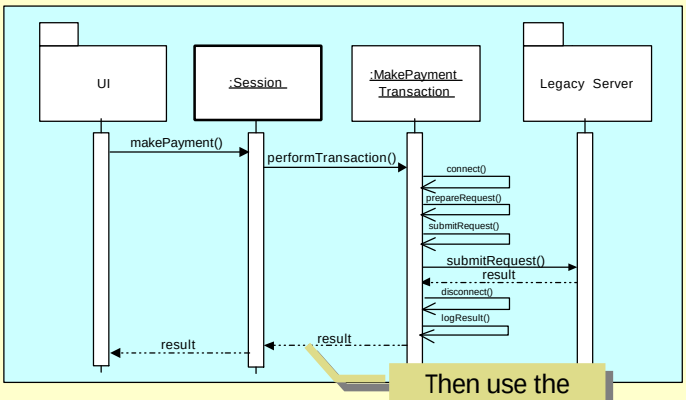
Use Multiple Descriptions

- Get comfortable looking at a design at different levels of abstraction. Describe a design multiple ways:
  - Use CRC cards to set the stage and explain object roles and responsibilities
  - Draw UML sequence diagrams to show specific interaction sequences
  - Write text to explain the details of objects' behavior during a specific interaction

Collaboration Storytelling Strategy

- Tell interesting stories—not every story is worth telling
- List the items you want it to cover
- Establish scope, depth and tone
- Decide how detailed the story should be
- Choose the appropriate forms—tell it, draw it, describe it
- Organize your story
- Revise, clarify, compress, and expand as needed
- Present your design so important things get emphasized

Illustrate The Non-Exceptional Case



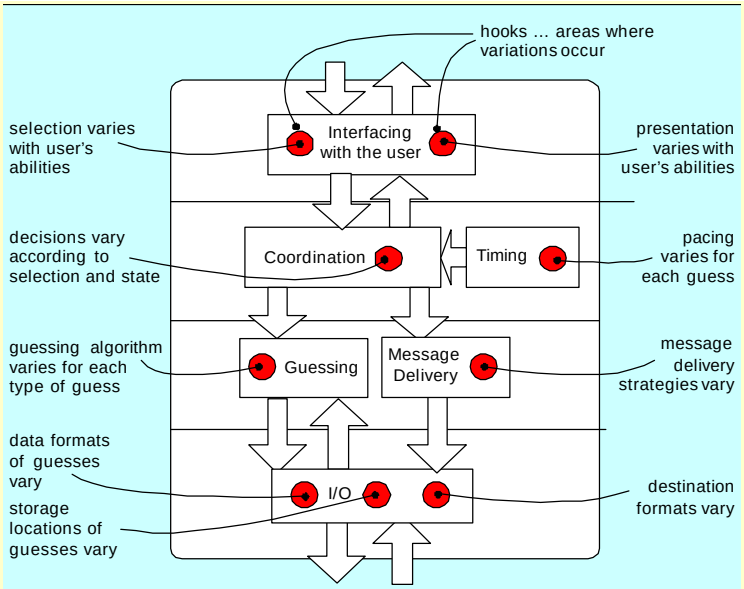
Then use the non-exceptional case to guide your consideration of exceptional cases.

Explain And Document Policies

Exception or Error	Recovery Action	Affect on User
Connection is dropped between UI and Domain Server after transaction request is issued.	Transaction continues to completion. Instead of notifying user of status, transaction is just logged. User will be notified of recent (unviewed) transaction results on next login.	User session is terminated… user could've caused this by closing his or her browser, or the system could have failed. User will be notified of transaction status the next time they access the system.
Connection dropped between domain server and backend bank access layer after request is issued.	Attempt to re-establish connection. If this fails, transaction results are logged as "pending" and the user is informed that the system is momentarily unavailable.	User will be logged off with a notice that system is temporarily unavailable and will learn of transaction status on next login.

Use descriptions approachable to marketers, developers and other stakeholders

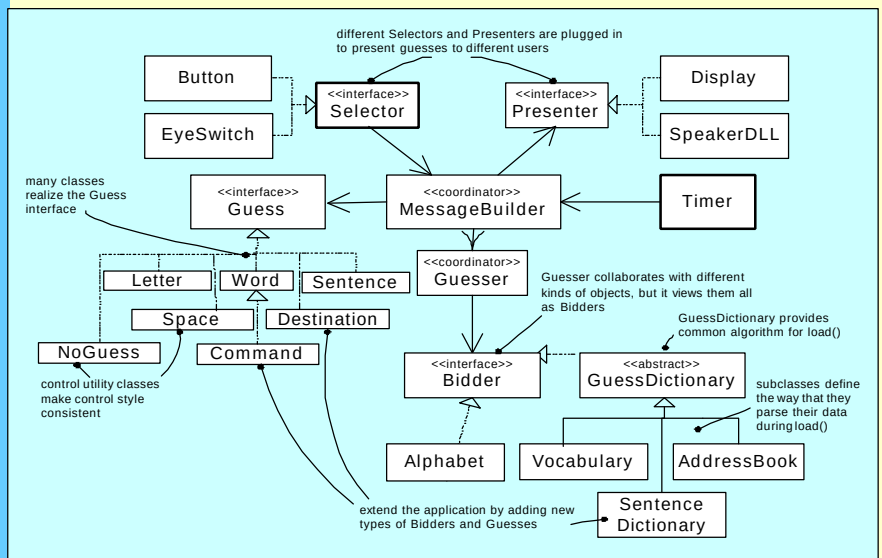
Consider Your Audience

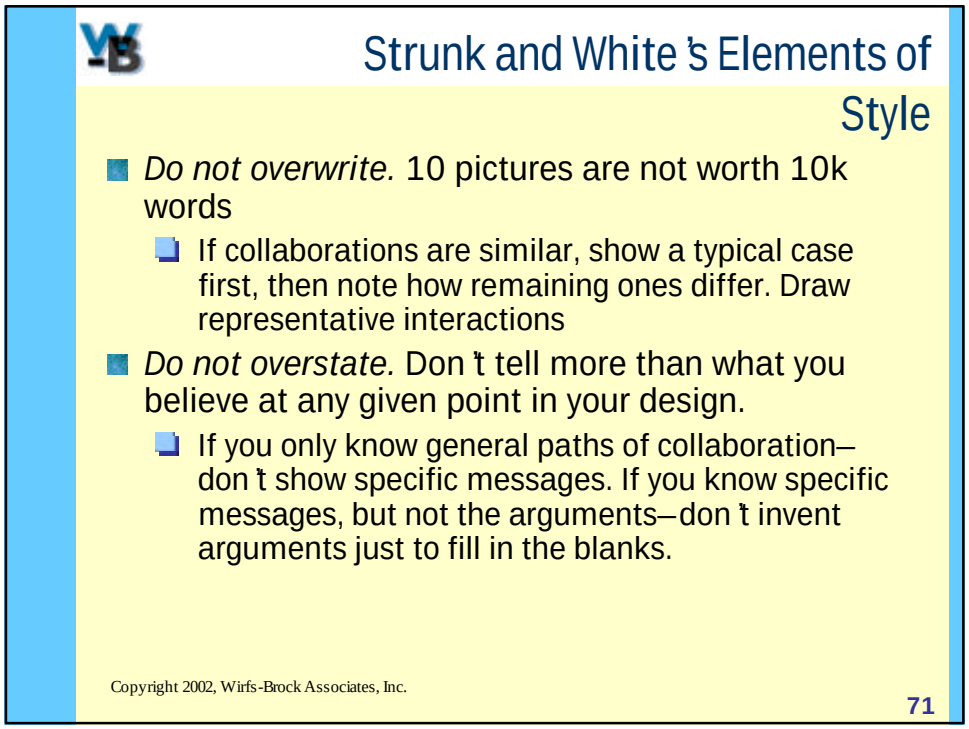


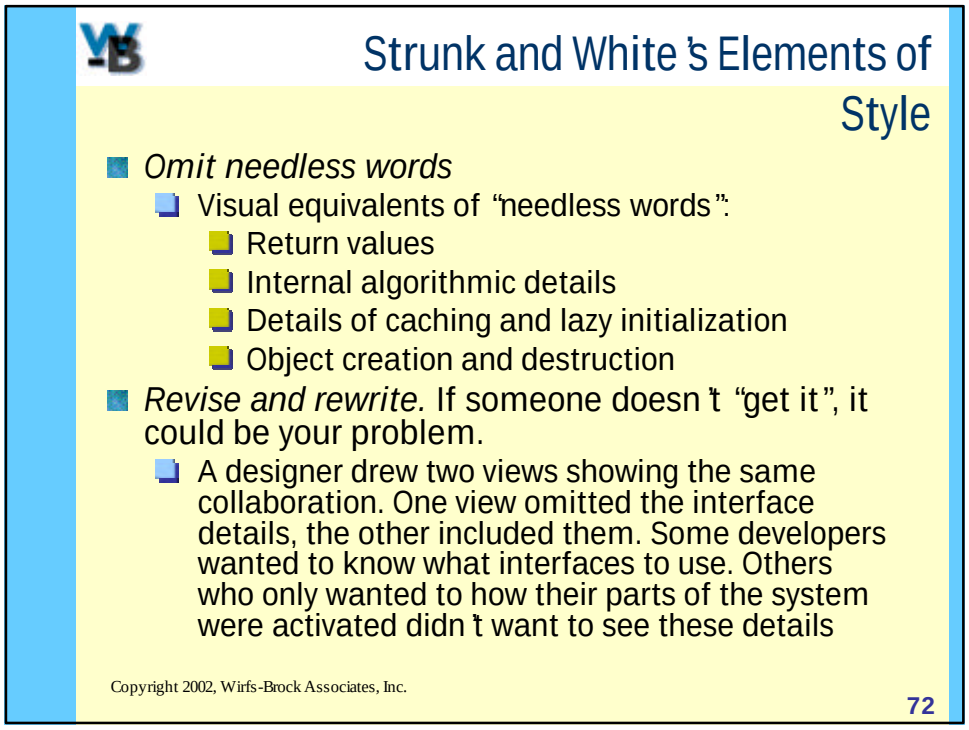
Copy

69

The Same View In UML




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
Do not overstate. Don't tell more than what you believe at any given point in your design.
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.
Copyright 2002, Wirfs-Brock Associates, Inc.
71


Strunk and White's Elements of Style
Omit needless words
Visual equivalents of "needless words":
Return values
Internal algorithmic details
Details of caching and lazy initialization
Object creation and destruction
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
Copyright 2002, Wirfs-Brock Associates, Inc.
72

Strunk and White's Elements of Style

- *Do not affect a breezy manner.* Don't intentionally 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 story. Get it down, then figure out how to show it.
- *Be clear.* Choose the right form of expression
  - To emphasize message ordering, use a sequence diagram. If timing is critical, add timing marks.
  - Add a running commentary to explain.

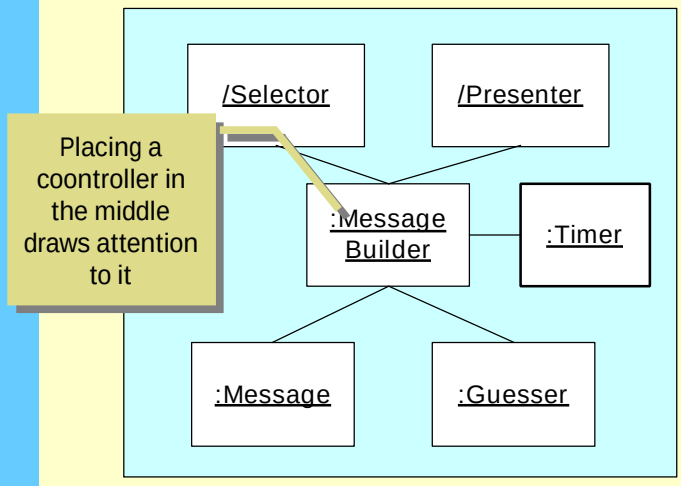
Strunk and White's Elements of Style

- *Make sure the reader knows who is speaking.* If you are telling a story from one perspective, stick to that storyline.
  - 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.
  - If your intent is to show how a complex responsibility is divided among collaborators, show what helper methods are invoked. Stop there.

Show Focused Interactions

- Decide what to emphasize!
- Consciously decide to ignore certain details
  - User interface
  - Delegated action
  - Details of an algorithm
- An object's central location, size and **boldness** give it emphasis

Visual Emphasis: Locate Important Objects in the Center



Place a controller in the middle. Messages radiate from it like spokes on a wheel.

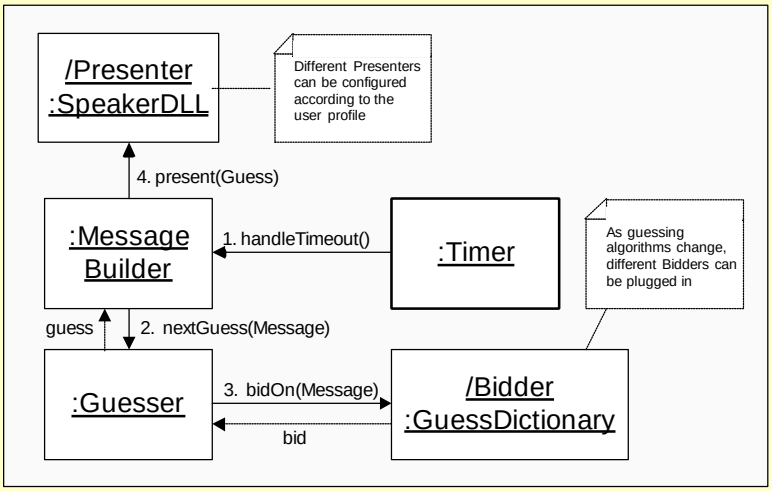
Being Detailed Isn't Always Better

- Precision should be a conscious decision
- Use UML when you want to be precise and rough sketches when you want to convey the gist of your design
- UML can be used with more or less precision. Add details to sequence diagrams if you need to show:
  - Branching (using guard expressions)
  - Iteration
  - Return values
  - Timing (using timing marks and expressions)
  - Object creation and destruction

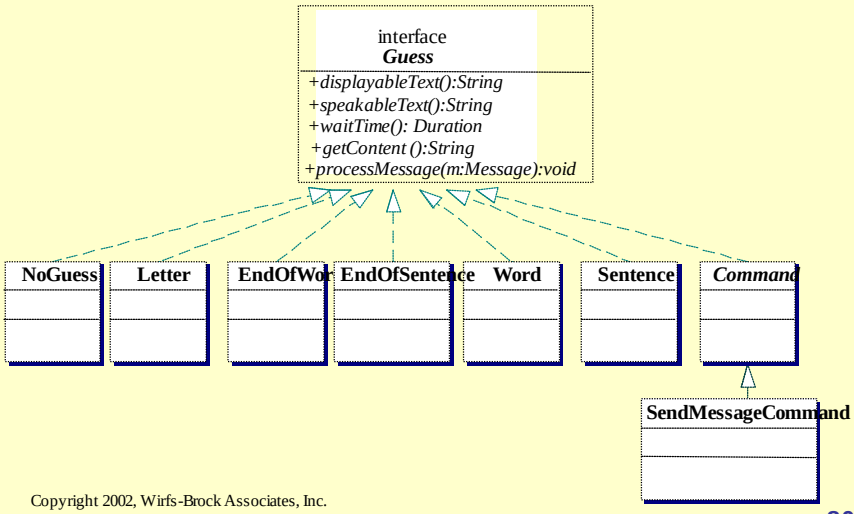
Show How to Adapt a Collaboration

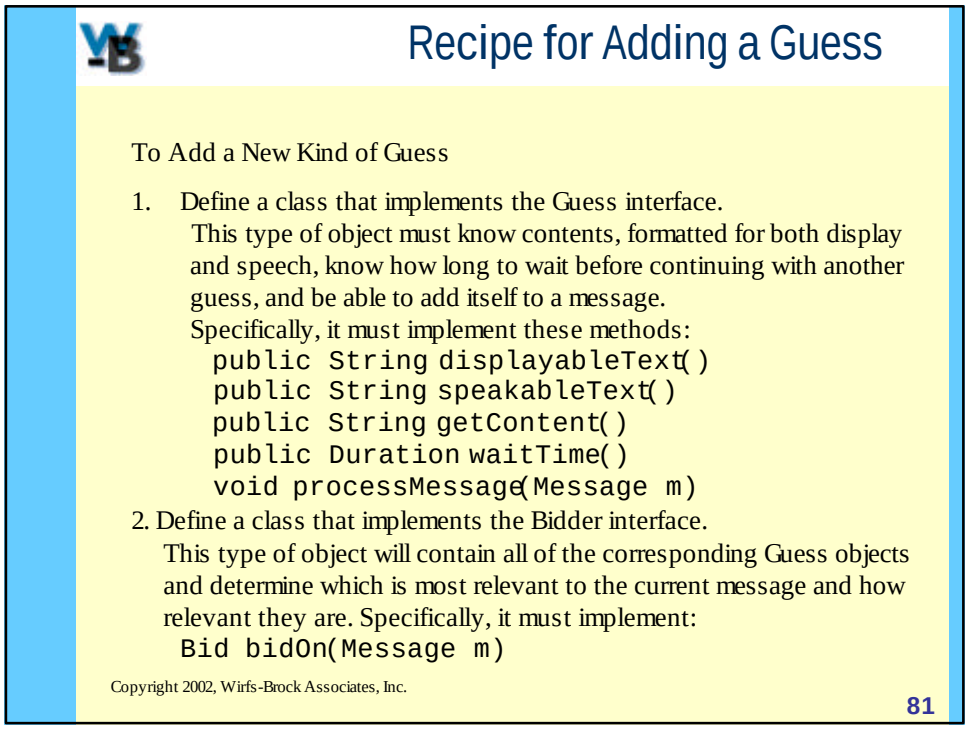
- To explain how to adapt a collaboration, you really need to explain three things:
  - the current design
  - what aspects are adaptable
  - how to make these adaptations
- Start by explaining how your design works
- Move on to showing how it can be modified
  - Show step-by-step procedure for adapting it
  - Add notes to "pluggable" objects

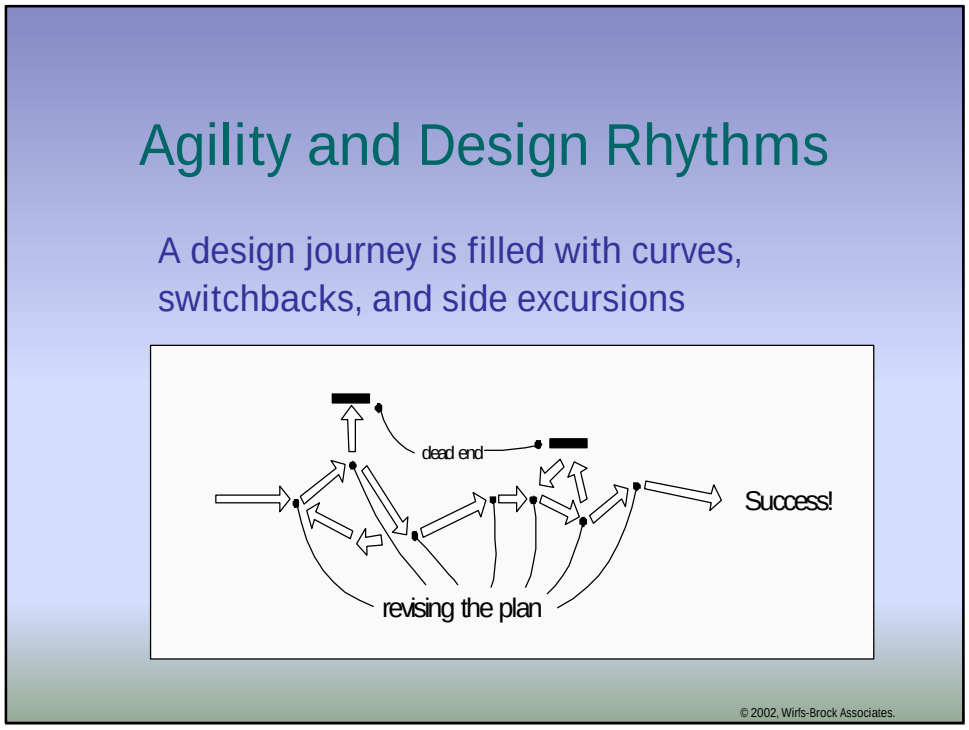
What Can Be Configured

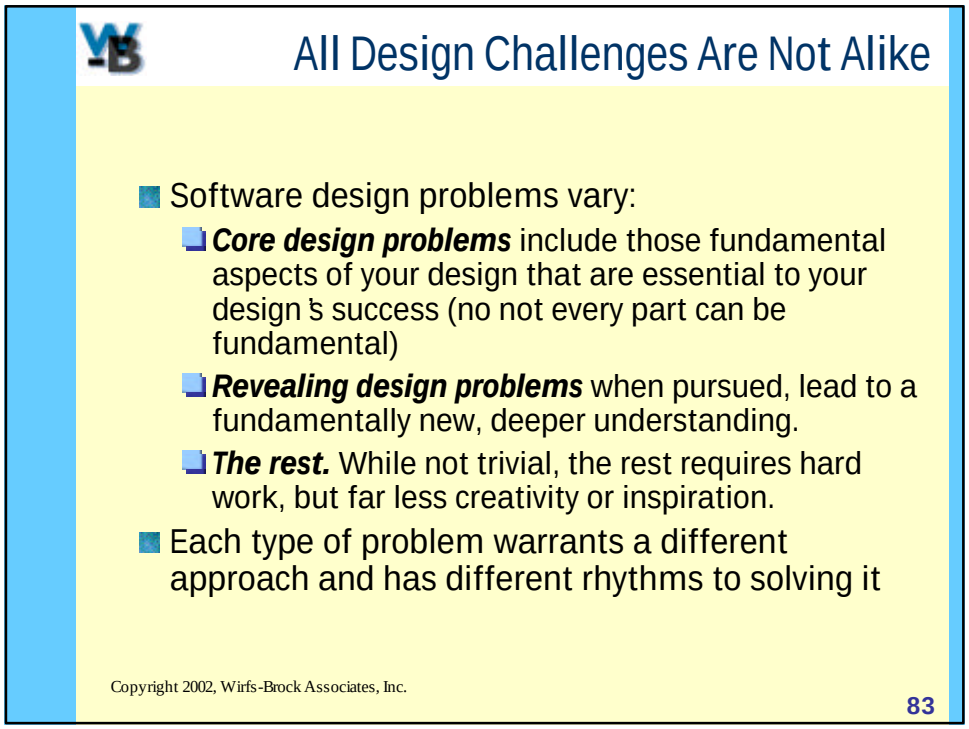


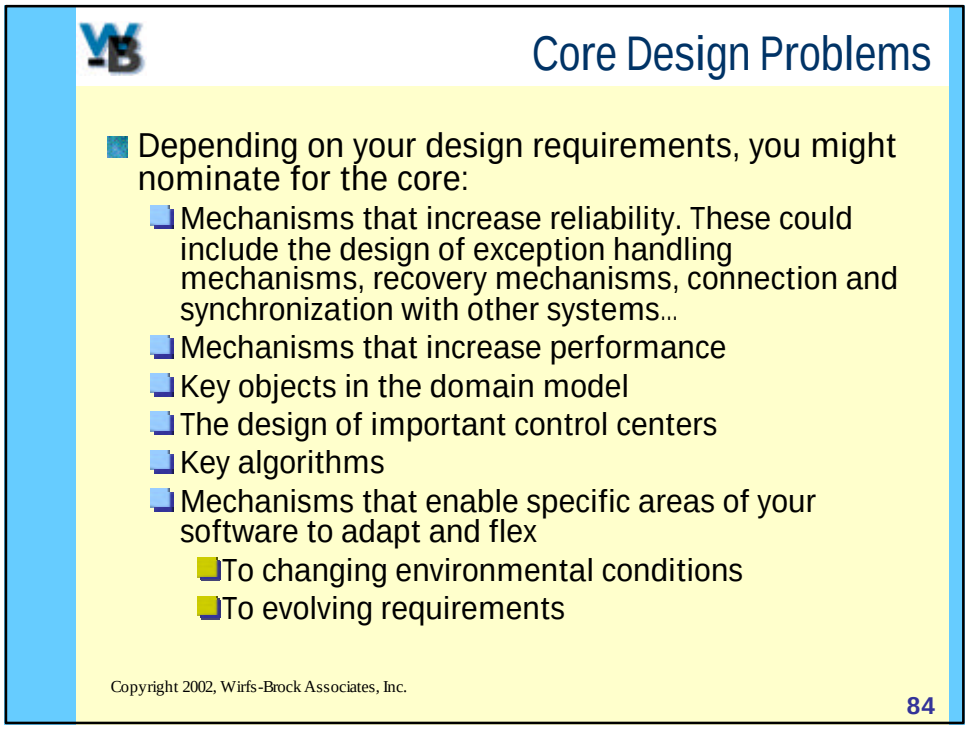
What Can Be Guessed




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:
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:
Bid bidOn(Message m)
Copyright 2002, Wirfs-Brock Associates, Inc.
81


Agility and Design Rhythms
A design journey is filled with curves, switchbacks, and side excursions
dead end
Success!
revising the plan
© 2002, Wirfs-Brock Associates.


All Design Challenges Are Not Alike
Software design problems vary:
Core design problems include those fundamental aspects of your design 's success (no not every part can be fundamental)
Revealing design problems when pursued, lead to a fundamentally new, deeper understanding.
The rest. While not trivial, the rest requires hard work, but far less creativity or inspiration.
Each type of problem warrants a different approach and has different rhythms to solving it
Copyright 2002, Wirfs-Brock Associates, Inc.
83


Core Design Problems
Depending on your design requirements, you might nominate for the core:
Mechanisms that increase reliability. These could include the design of exception handling mechanisms, recovery mechanisms, connection and synchronization with other systems...
Mechanisms that increase performance
Key objects in the domain model
The design of important control centers
Key algorithms
Mechanisms that enable specific areas of your software to adapt and flex
To changing environmental conditions
To evolving requirements
Copyright 2002, Wirfs-Brock Associates, Inc.
84

How Do You Decide What's Core?

- What are the consequences of "fudging" on that part of the design?
  - Would the project fail or other parts of your design be severely impacted? Then its definitely core.
  - When there are fundamentally different expectations, dig deeper. Someone may know something that others have ignored.
- Whether you classify something as part of the "core" or part of the "rest", you'll still have to deal with it—it's a matter of emphasis.
  - Give design tasks the attention they deserve and be clear on your priorities.

Revealing Design Problems

- Revealing design problems are always hard.
  - They may be hard because coming up with a solution is difficult—even though that solution may eventually be straightforward.
  - They may not have a simple, elegant solution.
  - They may not be solvable in a general fashion—each maddening detail may have to be tamed, one at a time.
  - They may require you to stretch your thinking and invent things that you have never before imagined.

Core Problems Can Be Revealing

- Not all core problems are revealing ones. Sometimes when you work on a core problem, you discover it to be a revealing one, too.
- What distinguishes revealing problems from core problems is their degree of difficulty and the element of surprise, discovery and invention.

Revealing Problems lead to deep understanding

- Solutions to revealing problems can touch on any aspect of a design:
  - They can impact an application's control architecture, key responsibilities of core objects, design of central services, complex algorithms or interfaces to external systems.
- They can cause you to completely shift your world view and discard what you had assumed to be a fundamental truth about your design, and replace it with something more complex.
- If you find yourself reacting, "Nah..that could never be!" to a design challenge, you just might have uncovered a revealing problem.

Some Problems Are *Really* Hard

- Wicked problems characteristics
  - They are hard to state concisely
  - They can be symptoms of other problems
  - They are one-of-a-kind
  - Solutions are open to value judgments
  - Solutions can be fuzzy or hard to describe
  - There is no obvious way to verify that a proposed solution fixes the problem
  - Their solutions have unforeseen consequences

. . . they are "tamed" not "solved"

Solving Wicked Problems

- Problem solving requires
  - the ability to shift your perspective and vary the problem,
  - the ability to gauge whether an approach, if pursued, is likely to bear fruit, and
  - knowing when you've hit a dead end
- Vary your perspective by
  - generalizing, specializing, forming analogies, decomposing and recombining aspects

Example: Redefining the Problem

- In Java, C#, or Smalltalk, memory is automatically recovered from objects that are no longer used. Early implementations used reference counting to manage memory. This technique is simple, but very expensive. To speed up garbage collection algorithms, implementers of the languages redefined the problem—and now use sophisticated scavenging algorithms

Often, redefining a problem doesn't simplify; it just opens up new possibilities

Observations On Solving Wicked Problems

- Time is required to let things "soak in" and to form connections between the problem and past experiences
- Wicked problems are always either squarely demanding your undivided attention or lurking in your background thoughts
- They cannot be solved by a committee, although a proposed solution can be reviewed and tuned by a group
- It can be nearly impossible to predict when they will be solved

Handling The Rest

- The rest is what you work on day in and day out when nothing else demands your attention. What might be included:
  - Common error logging or reporting mechanisms
  - Data conversion
  - Basic features that are similar to ones you've already implemented
  - Optional features
  - Alternate strategies for accomplishing some behavior

Sorting Out The Rest From The Core

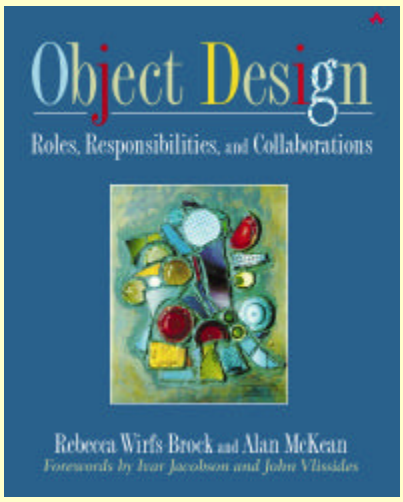
- It's easy to get caught up in a debate of what's core and what's in the rest. Don't waste your time.
- If you know that something is just basic design work that has to be there, nothing special, nothing fancy, it's probably part of the rest.
  - What about exception handling? Why isn't the 90% of your design work that supports the unhappy path scenarios a core design task? Well, depending on your project, they might be. Or they might not.
- Core problems should be given more attention. That doesn't mean the rest gets slighted. It just isn't at the top of your list.

Agility Means Finding a Balance

Those who pursue agile development practices, "seek to restore credibility to the concept of methodology. We want to restore a balance. We accept modeling, but not in order to file some diagram in a dusty corporate repository. We accept documentation, but not hundreds of pages of never-maintained and rarely used tomes. We plan, but recognize the limits of planning in a turbulent environment."

---Jim Highsmith

Resources



- Read more about seeing and thinking, wicked problems, and object design strategies in our new book

  Object Design: Roles, Responsibilities and Collaborations, Rebecca Wirfs-Brock and Alan McKean, Addison-Wesley, 2003
- www.wirfs-brock.com for articles & resources