Introducing Problem Frames

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

Problem Frames: A tool for seeing typical patterns of software tasks

frame—a structure that gives shape or support
A Problem Frame...

“... defines the shape of a problem by capturing the characteristics and interconnections of the parts of the world it is concerned with, and the concerns and difficulties that are likely to arise. So problem frames help you to focus on the problem, instead of drifting into inventing a solution.”

—Michael Jackson
Why frame problems?

“It’s not easy to focus on problems in software development.”

Problems:
Partly precise, partly imprecise

Scylla
Programming: Precise

and

Charybdis
Sociology, psychology: Imprecise

A case study: Email
Framing strategy: divide and conquer

- Decompose problems
- Focus on the requirements and the concerns of each subproblem

Tactics for decomposing problems

- Identify the core problem
  - Sending and receiving mail
- Look for ancillary problems
  - Constructing mail
  - Managing mail folders
  - Sorting mail
  - Reading mail
  - Maintaining address lists
- Examine problem concerns for more
Look for different tempos

- When parts of a problem have different speeds
  - Expect to treat them as different subproblems
  - A shared domain is often dynamic in the faster subproblem and static in the slower one

Mail folders and email messages

Jackson’s view of a context diagram
The context diagram relabeled

Domain types:
Causal—predictable cause & effect
Biddable—usually people
Lexical—symbolic values

Problem Diagrams:
Simple workpieces problem

a tool that allows users to create and manipulate structures

shared phenomena interfaces

Email Editing Tool

Email messages

User

Correct effects of user’s commands on message contents

"machine" domain

biddable domain

lexical domain

requirement
Transformation problem

Converting some input to some output according to certain rules

- Email Decoder
- Encoded Email
- Decoding Requirements
- Viewable Email

Information display problem

Information is needed about some thing's state and/or behavior

- Incoming Mail
- Junk Mail Filter
- Identify Junk Mail Requirement
- Filter Report
Commanded behavior problem

controlling the behavior of some “thing” according to operator commands

Required behavior problem

controlling behavior of some “thing”
Jackson’s view on problem descriptions

- Optative- describes domain properties and behavior that your software must guarantee
- Indicative- describes objective truth or facts

About phenomena and domains

- Individual phenomena-
  - Event-a happening: MailSent
  - Entity-an individual that persists over time and changes its properties and states: EmailMessage, Inbox, Mailfolder
  - Value- a quantity or quality: 50 junk mail confidence rating
Relating phenomena

- State-relating values to entities
  \textit{JunkMailRating(EmailMessagexxx, 100)}
- Roles-relation between an event and individuals \textit{MailMoved(Inbox, ToDoFolder, EmailMessageyyy)}
- Cause and effect \textit{MailRecieved(Inbox, EmailMessagexxx) \rightarrow JunkMailRating(EmailMessagexxx, 100)}

---

Shared Phenomena

Interfaces between domains are "direct": Events, states, and values are shared between connected domains
Required behavior problem frame

A “frame diagram” - a fancier generic problem diagram with named parts and sets of phenomena

Required behavior concerns

1. We will build our software to behave like this so that...
   {specification}

2. Knowing that the controlled domain works like this...
   {domain description}

3. We'll be sure it acts this way
   {requirements}
Stylized Required Behavior Questions

- What external state must be controlled?
- How does your software find out whether its actions have had the intended effect?
- What should happen when things get “out of synch”?
- How and when does your software decide what actions to initiate?
- Is there a sequence to these actions?
- Are there complex interactions with your software and the thing under its control?

Workpieces frame concerns
Stylized Workpiece Questions

- Will it take different forms?
- Does it have an interesting lifecycle (or is it just something that is changed and then treated as "static" after each change?)
- Is it passed around between various users? Is there a workflow associated with a workpiece?
- Should it persist? Should it be published or printed?
Stylized Transformation Questions

- What data do you start with?
- How will it be changed?
- Is the transformation complex?
- Will it always work? What should happen when you encounter errors in the input?
- Is the transformation “lossy” or reversible?
- What speed, space, or time tradeoffs are there?

Information display concerns

...the software will detect these phenomena...
(specification)

...then because the things are like this...
(domain properties-a)

...when the state of the world is this...
(requirement)

Display - real world

Your Software Information Machine

Display

...and cause these events...
(specification)

To the output will be this...
(domain properties-b)

...which correspond as required to what is happening in the world...
(requirement)
Stylized Information Display

Questions

☐ How precise does the information need to be? Is the information “fuzzy”?
☐ How much computation does your software have to do to come to an observation?
☐ Is the user only interested in current information? Or is historical information important?
☐ Are there questions that the user may want to ask about the information? What are they? How easy are they to accurately answer?
☐ Does you need to about it?
Stylized Commanded Behavior Questions

- What does the user need to know in order to “command” the system to do things?
- Do certain commands need to be inhibited? Do they always make sense?
- Is there a lag between issuing a command and the system performing the action?
- What happens when a command fails?
- Should certain commands be ignored?
- Do commands need to be reversible? logged? monitored or otherwise tracked?

Fitting problems to frames

The wrong frame will have some descriptions that don’t make sense and will leave out other necessary ones
Problem frame expectations

- For each frame there is a clear expectation of the relative complexities

<table>
<thead>
<tr>
<th>Frame</th>
<th>Complex</th>
<th>Simple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required behavior</td>
<td>Controlled domain</td>
<td></td>
</tr>
<tr>
<td>Commanded behavior</td>
<td>Controlled domain</td>
<td>Operator domain</td>
</tr>
<tr>
<td>Information display</td>
<td>Real world</td>
<td>Display domain</td>
</tr>
<tr>
<td>Simple Workpieces</td>
<td>Work pieces domain</td>
<td>User domain</td>
</tr>
<tr>
<td>Transformation</td>
<td>Input and Output domains</td>
<td></td>
</tr>
</tbody>
</table>

Frame Variants: Required behavior problem revisited

controlling behavior of some “thing”
The ideal...

☐ Your software directly shares phenomena with the problem domain

☐ A rich interface gives access to phenomena it needs to detect or control

Reality...

☐ An “intermediary” often lies between

☐ This connection domain can be quirky
Sometimes documents are connection domains

- Identify people
- Bar coded labels
- Bank cards in ATMs
- PGP Certificates of trust

Composite problems

- Interactions among subproblems gives rise to composition concerns
  - Consistency
  - Precedence
  - Interference
  - Synchronization
**Commanded and required behavior**

- **Consistency:** Is it possible to satisfy both requirements?
- **Precedence:** Where requirements aren’t consistent, which takes precedence?

**Integrating framing with other req’ts**

- desired behavior and given domain properties can be missing or jumbled

Framing can help clarify
### Requirements Roadmap

<table>
<thead>
<tr>
<th>Business Requirements</th>
<th>User Requirements</th>
<th>Software Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Who?</strong></td>
<td>Stakeholder Categories</td>
<td>Prototypes Dialog Hierarchies Personas</td>
</tr>
<tr>
<td><strong>What?</strong></td>
<td>User Requirements Table*</td>
<td>Class Model Data Dictionary Data Tables</td>
</tr>
<tr>
<td><strong>When?</strong></td>
<td>Context Diagram</td>
<td>State-Data Matrix</td>
</tr>
<tr>
<td><strong>Why?</strong></td>
<td>Business Policies</td>
<td>Decision Tables Decision Trees</td>
</tr>
<tr>
<td><strong>How?</strong></td>
<td>Process Map*</td>
<td>Use Case Map Use Case Packages</td>
</tr>
</tbody>
</table>

* Business Model

Source: Ellen Gottlendiener, EBG Consulting

Where problem frames don’t fit

- Framing’s emphasis on phenomena doesn’t help describe or understand:
  - Mathematical computations or algorithms
  - Graphics user interfaces
  - Compilers
  - ...

Wirfs-Brock Associates  www.wirfs-brock.com  Copyright 2006
What about a more agile world?

- Jackson advocates fully understanding problems before starting design
- Agile developers expect to incrementally discover requirements. Framing
  - ...can focus design spikes
  - ...can lead to deeper understanding of user stories and their dependencies

How do Problem Frames Relate to Sprints and Story Cards?

![Diagram showing the relationship between problem frames and sprints]

ECI: GetEmailToSend from FM1 EmailToBeSent

Internet Service Provider

Automatic Email Controller

Email transfer schedule

Mail Folder Manager

Send and gueshock for Email or predefined schedule
How do frames relate to Sprint Story Cards?

- Implementation of feature “simple path” (v 0.1)
- Implementation of feature extended case (v 0.2)
- Implementation of feature extended case (v 0.3)
- Technical req’t story
- Designed Domain affecting story
- Domain affecting story

Resources

- Problem Frames: Analyzing and structuring software development problems, Michael Jackson, Addison-Wesley, 2001
- Study guide (questions/key ideas) for Problem Frames
- Software Requirements And Specifications, Michael Jackson, Addison-Wesley, 1995
- A website devoted to problem frames: http://www.ferg.org/pfa/
- Michael Jackson’s home page: http://mcs.open.ac.uk/mj665/
- Some of my blog entries: http://www.wirfs-brock.com/rebeccasblog.html
**Introducing Problem Frames**

Rebecca Wirfs-Brock
Wirfs-Brock Associates,  [www.wirfs-brock.com](http://www.wirfs-brock.com)

**Problem frames** are a way of mentally dividing our software’s purpose into manageable chunks. Software systems can be thought of as a set of sub-problems or “problem frames”. By breaking down a problem into its constituent problems, you can consider a large system one smaller piece at a time. Michael Jackson, who invented the notion of problem frames, writes about them in *Problem Frames: Analyzing and structuring software development problems*. Jackson suggests that because software serves so many purposes that software developers start by describing and structuring their problems in a way that, according to Jackson, is “rarely necessary in other engineering disciplines, where the diversity of problems to be solved is much smaller.” As software analysts, developers or agile designers must become adept at asking: What kind of problem is this? What is our software all about? What purpose does it serve? What behavior and properties must our software have to achieve that purpose?

Each different class of problem frame has specific concerns and issues. When you think about a problem, if you can “fit your problem (or a piece of it) into a relevant frame” then it will lead you to ask appropriate questions and make appropriate tradeoffs. Here are definitions of Jackson’s five problem types or frames and example frame diagrams:

- **Required behavior**—controlling state changes of something outside your software machinery according to specific requirements.

![Diagram of Automatic Email Controller](image1)

- **Commanded behavior**—controlling changes based on an operator or user’s commands.

![Diagram of Commanded Behavior](image2)
• Information display—produce information about some observable phenomena

Frame diagrams are just a convenient iconic way to represent the structure of a problem, but they are secondary to the real value of framing—a tool to help you gain understanding and focus requirements and design activities. If you understand the nature of the problem your software needs to address, you can ask relevant questions that help shape and focus your work. You can use problem framing in many different situations even if you don’t adopt Jackson’s formal approach.

• Simple workpieces—a tool that allows users to create and manipulate structures, so that they can be copied, printed, analyzed, or used

• Transformation—convert input to one or more outputs according to specific requirements

To enhance requirements descriptions and artifacts with richer descriptions. For example, if you are aware that a connection domain sits between your software and a “controlled domain”, your requirements can be beefed to describe exceptional conditions and how they should be addressed and possibly, to model that connection domain’s properties.
To distinguish truths or facts (indicative qualities) from desired (optative) behavior which is often imprecisely expressed as statements beginning with “shall” or “should”.

To help you recognize where state models are appropriate to more precisely model complex behaviors.

- In agile development:
  - To initially brainstorm what kinds of design challenges will predominate and what parts of your software they are likely to impact. In all but the simplest system there are usually multiple problems (and frames) that are evident. Problem framing is a good way to get teammates acquainted with upcoming design work and identify the potentially hard parts.
  - As you discuss specific user stories with your customer. While I don’t even mention problem frames to customers, I keep them in mind as we discussing any issue. I use them as a mental tool to sharpen my thinking. If you think about which problem frame is relevant (and what concerns there are) you will find yourself asking questions that buy you more information. And you can have more meaningful discussions with your customer about what your software should or shouldn’t do.
  - To assess additional work during a design spike. As you dig deeper into implementation you need to rethink and occasionally reframe the problems you are solving. A design spike happens whenever something is more complex than you had thought. It could be that reframing the problem might bring clarity.

Frame Concerns. Your goal is to design and build software that will behave appropriately and solve the customer’s problem. Jackson advocates that you convince yourself and your customer that your proposed software will tackle the right problem by writing an appropriate set of descriptions about the problem domains. As a problem framer, your central task is to investigate and describe problem domain properties. Each class of frame has a different set of concerns that are typically addressed.

This is one area where Jackson and agile developers diverge on their approaches (and value equation). While I may advocate for formal descriptions when they add value, I find Jackson’s insistence on writing descriptions of various domain properties to be a difficult task for most developers whether they are agile or not. I find these formalisms to be less valuable than knowing what questions to ask and what issues are commonly encountered in particular problem frames. So instead of going formal, I find myself asking probing questions about a particular frame. Once I’ve framed a problem, I can start asking questions. Or conversely, as I am asking questions I’m exploring what frames seem to fit and push harder to gather appropriate requirements.

Here are some stylized questions to ask about a workpiece frame:
- What are the basic elements of the workpiece?
- Will it take different forms?
- Does it need to be shared? If so, how?
- Does it have an interesting lifecycle (or is it just something that is changed and then treated as “static” after each change?)
- Is it passed around between various users? Is there a workflow associated with a workpiece?
- Should it persist? In what forms? Should it be published or printed?

Here are some questions to ask about required behavior problems:
- What external state must be controlled?
- How does your software find out whether its actions have had the intended effect? Does it need to know for certain, or can it just react later (when the state of some thing is not as expected)? What should happen when things get “out of synch” between your software and the thing it is supposedly controlling?
- How and when does your software decide what actions to initiate?
- Is there a sequence to these actions? Do they depend on each other?
- Are there complex interactions with your software and the thing under its control?
- Can you view the connection between your software and the thing under control as being direct (easier) or do you have to consider that it is connected to something that transmits requests to the
thing being controlled (and that this connection can cause quirky, interesting behavior)? If so, then you may need to understand the properties of this “connection domain” that stands between your software and the thing being controlled?

Here are some questions to ask about transformation problems:

- What data do you start with?
- How will it be changed?
- Is the transformation complex?
- Will it always work? What should happen when you encounter errors in the input?
- Is the transformation “lossy” or reversible?
- What speed, space, or time tradeoffs are there for performing any transformation?

Here are some questions to ask about commanded behavior problems (in truth these are only the tip of the iceberg):

- What’s a good model of user-system interaction?
- What does the user need to know in order to “command” the system to do things?
- Do certain commands need to be inhibited based on the current state of the system? Do they always make sense? Does a sequence of actions make sense?
- Is there a lag between issuing a command and the system performing the action? Is that a problem?
- What happens when a command fails? How should users be involved in “steering” the software when a command fails?
- Should certain commands be ignored (e.g. how many times do you need to press the elevator button to call the elevator to your floor)?
- Do commands need to be reversible? logged? monitored or otherwise tracked?

Here are some questions to ask about information problems:

- What is the form of “observation” that the software must make about some event or fact or thing? Is it difficult to ascertain when an event has occurred? (For example, if your software is trying to record how many “vehicles” passed over sensors placed on the road it may be very difficult to characterize what constitutes a vehicle—is it two axles passing within a time period, but what about motorcycles, backed up slow traffic, etc., etc.)?
- How precise does the information need to be? Is the information “fuzzy”?
- How much computation does your software have to do to come to an observation? (For example, consider assigning a “junk mail rating” to an email, based on Bayesian analysis of the contents of the current message based on sample data currently loaded into the junk mail box?)
- Is the user only interested in current information? Or is historical information important?
- Are there questions that the user may want to ask about the information? What are they? How easy are they to accurately answer?
- Does your software need to construct a “model” of the phenomena being observed in order to answer questions about it?

**Additional Resources**

Two books by Michael Jackson

*Problem Frames: Analyzing and structuring software development problems*, Michael Jackson, Addison-Wesley, 2001

*Software Requirements and Specifications*, Michael Jackson, Addison-Wesley, 1995

A website devoted to problem frames and their application: http://www.ferg.org/pfa/

Jackson’s home page: http://mcs.open.ac.uk/mj665/

Some of my blog entries are about framing: http://www.wirfs-brock.com/rebeccasblog.html

If you are interested in a copy of some study questions for Jackson’s book or wish to pursue practical hands on framing activities contact me at rebecca@wirfs-brock.com

© 2006, Wirfs-Brock Associates