The General and the Turkey: a Participatory Introduction to Large-Scale Software

Mary Shaw, Carnegie Mellon University
The General and the Turkey:  
A Participatory Introduction to Large-Scale Software  

Mary Shaw  
Computer Science Department  
Carnegie Mellon University  
Pittsburgh PA, USA  
Internet: mary.shaw@cs.cmu.edu  


Software engineering education includes not only the education of software engineers but also the education of others about software engineering. From time to time we are called upon to explain “software engineering” to audiences that have never considered the organization or construction of large software systems or dealt with programs larger than a few pages. With little audience experience to build on, this can be a daunting task.

This educational item engages the audience in simulated operation of a large system. A dozen volunteers play the parts of databases, communication protocols, and human users in a series of transactions that lead quite innocently to a surprising transaction that actually took place in real life. I have used it with university freshmen; it can potentially be used with groups of high-school students or adults who are not computer professionals (and who are uninhibited enough to participate). The preferred group size is 15 to 40.

The activity addresses the problem of explaining how large-scale software is constructed from components and what sorts of things can go wrong. It also touches on system integration and evolution, requirements analysis, the software development process, and discrepancies between the users’ and the designers’ model of a system.

Keywords: software engineering education, audience participation, introduction to software system organization for nontechnical audience, teaching technique, sample lecture

Introduction

From time to time we are called upon to explain “software engineering” to students with little programming background, to professionals in other fields, or to the lay public. These groups may have experience as computer users, but they have little direct experience with programs as such and they typically have no experience with multi-module systems, with modification and maintenance, with subtleties of requirements analysis, or with processes associated with large-team software development.

This “educational entity” addresses part of that task: it shows how a software system is decomposed into individual components and it shows how correct operation is sensitive to the interfaces and interactions of the components. The entity is a simulation of a data processing task based on audience participation. It leads the audience on a tour of a distributed database system, but there is no special emphasis on database operation. This setting was chosen because the required functionality is easy to understand and the individual components are easy to explain. In order to hold the attention of the audience, the example itself is based on an incident with a surprising outcome that actually took place.
Particulars

The objective of the exercise is to show the class or audience:

- how a large system is built up from individual components or subsystems
- the implications of making choices between design alternatives
- the consequences of even slightly mismatched module specifications
- the sorts of discrepancies that arise between the designer’s view of the system and what the users actually do

I designed the exercise for use with university freshmen. I have classroom tested and revised it with that audience. It is also suitable for other audiences with minimal knowledge of computing -- perhaps just what a program is, plus intuition about the representation of data in tables. The audience should be sufficiently relaxed (or unselfconscious) to be willing to participate. The group size should be between fifteen and forty.

The format of the exercise is a participatory simulation of a distributed inventory database system. The instructor has 12 tasks for volunteers; each either simulates a human user or a component of the system. The instructor leads the audience through a sequence of individual simple steps, beginning with a simple request for goods and culminating with a distributed query.

The incident that underlies the example took place during the Desert Shield operation in the Middle East. Shortly before Thanksgiving (an American holiday for which the traditional meal is roast turkey), a General wanted to make sure that each American serviceman or woman would receive a traditional holiday dinner. He asked for the number of turkey meals-ready-to-eat on hand; the warehouse database system responded with a very large negative number. I slightly modified the outcome and created a simplified database model that allows the exercise to be carried out with novices. The details of this example are distinctly American. This is fine for an American class, as it gives the students a real event to relate to. It should be easy enough to change it for use in some other country.

The Script and Supporting Materials

The exercise uses one sheet of instructions for each volunteer. These sheets take the form of overhead projection transparencies. The volunteer keeps the sheet as a reminder of what he or she has volunteered for and a place to keep track of changing information. To keep the audience in touch with proceedings, the instructor uses the corresponding overhead projection transparency each time the volunteer is active. These instructions/transparencies appear below.

The detailed script of the exercise includes

- Instructions to be read to the volunteer
- An indication of the action the instructor is trying to elicit from the volunteer
- Notes on design issues the instructor can bring out at each stage

This script appears as annotations on the instructions/transparencies. The entire set appears as the body of this report.

Acknowledgments

I based this exercise on an actual incident reported to me by Al Despain (University of California, Berkeley) during a discussion of data fusion. I modified the incident slightly and worked up the participatory activity in response to a request from Raj Reddy for a contribution to “How Things Work”, his introduction to computing for freshmen. Carnegie Mellon’s School of Computer Science provided support.
Desert Shield
Quartermaster Exercise

Introduction to large software systems
   How they’re organized
   Consequences of inconsistencies
   Design choices
Based on real-life example
   Totally, radically oversimplified
   But still captures essence of the problem

Script
This exercise is designed to show you how a large system is built up from individual subsystems. Further, it is designed to show some of the effects of building a big system out of components from different sources. We’ll need 12 volunteers along the way.

The setting is Desert Shield, a few years ago, in autumn. The Forces of Good are sitting in the desert hoping that the Forces of Evil will give in before anyone does anything drastic. We’re going to look at a supply problem.

Notes
This simply gets you started and gets everyone settled down. Don’t try to line up all the volunteers at once; pick them out as you go. This is in part practical: it may be hard to get the required number of volunteers at the outset, but you can hand single instruction sheets to individuals as you go along.
Quartermaster Unit 2467

Your job: keep the unit supplied with whatever goods it needs
You’ll make requests in the form
“"I need six camouflage shirts, size XXS”"

Volunteer 1: Quartermaster Unit 2467

Script
You are responsible for supplying your unit of 1000 people. You’re running short of size XXS camouflage shirts, and since Thanksgiving is coming you’d better order some turkey dinners.

Notes
This volunteer plays a human. It isn’t clear yet who’s listening to requests -- that comes in the next two stages.
The surprise at the end depends on getting the counts of turkey dinners to come out to the amounts on slides you have already prepared. Students sometimes get silly and order excess; in this case point out that they have to store the stuff. Later on we’ll have the quartermasters hedge their bets by ordering extra turkey dinner; if this volunteer tries to order extras now, offer praise for foresight and argue that it would be *even* better to order from another source (an opportunity that will arise later).
## Warehouse West Database

Keep accurate list of what’s in the warehouse and to whom it’s been committed

<table>
<thead>
<tr>
<th>Count</th>
<th>Stock #</th>
<th>Description</th>
<th>Recipient</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>4G9-23</td>
<td>1 ea Humvee Transmission</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4G9-23</td>
<td>1 ea Humvee Transmission</td>
<td>Unit 2467</td>
</tr>
<tr>
<td>2</td>
<td>4G9-23</td>
<td>1 ea Humvee Transmission</td>
<td>Unit 4692</td>
</tr>
<tr>
<td>25</td>
<td>SW6-51</td>
<td>100 ea Turkey MRE</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SW6-51</td>
<td>100 ea Turkey MRE</td>
<td>Unit 7491</td>
</tr>
<tr>
<td>10</td>
<td>SW6-51</td>
<td>100 ea Turkey MRE</td>
<td>Unit 9215</td>
</tr>
<tr>
<td>427</td>
<td>A3M-98</td>
<td>12 ea Camo shirts (XXL)</td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>W4K-98</td>
<td>8 ea Camo shirts (XXS)</td>
<td></td>
</tr>
</tbody>
</table>

Accept requests in same format as database

---

### Volunteer 2: Warehouse West

#### Script

You are the database for a supply warehouse. You’re responsible for keeping track of what’s in the warehouse and to whom it’s been promised. The information about what’s there is common inventory information, since it’s easier to ask the database than to go out and count Humvee transmissions. The information about what’s been promised helps load trucks sensibly.

Quartermaster 2467, do you see a solution to your problem? Well, almost. How do you get your request for shirts and dinners to the database?

#### Notes

Each kind of item in stock can appear many times in the database: once for each shipment promised and once for the unpromised remainder. When an order is accepted, the count for the unpromised amount must be updated when the new promise is entered. This is an opportunity to introduce the notion of a transaction.
Volunteer 3: Warehouse West Communication Protocol

Script

You're responsible for accepting information from quartermasters, transmitting it reliably to Warehouse West, and reformatting it in the format Warehouse West expects.

OK, Quartermaster 2467, tell WW Communications what you need. WW Communications, transmit this to Warehouse West. Warehouse West, update your database.

<pause> OK -- who's responsible for converting quartermaster counts to warehouse package counts?

<pause> Ah ... is anybody going to bother with a confirming message?

Notes

Point out that the warehouse counts warehousing units -- boxes, pallets, or whatever, whereas quartermasters count the number of individual items they need.

Discuss where knowledge of number of individual units per warehouse container should reside; this leads to discussion of pros and cons of protocol or warehouse doing the conversion. Someone other than the unit quartermaster is responsible for rounding the quartermaster's counts up to complete warehouse units.

This is also an opportunity to discuss the differences between batch and interactive orders, especially for error reporting. What should happen if the warehouse doesn't stock a requested item or if it doesn't have enough to satisfy the request?
Quartermaster Unit 4692

Your job: keep the unit supplied with whatever goods it needs
You'll make requests in the form
"I need thirty-two ping-pong tables"

Volunteer 4: Quartermaster Unit 4692

Script
You are responsible for supplying your unit of 1437 people. Your morale officer is about to sponsor a ping-pong tournament, but you don't have any ping-pong tables. Oh, yes -- Thanksgiving is coming over on this side of the desert, too. However, you're closer to Warehouse East.

Notes
Nothing special here yet.
Warehouse East Database

Keep accurate list of what’s in the warehouse

<table>
<thead>
<tr>
<th>Count</th>
<th>Description</th>
<th>Stock #</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>1 ea Ping-pong table</td>
<td>34-F78</td>
</tr>
<tr>
<td>1384</td>
<td>1000 ea Ping pong balls</td>
<td>87-3DT</td>
</tr>
<tr>
<td>18</td>
<td>1 ea Humvee Transmission</td>
<td>23-4G9</td>
</tr>
<tr>
<td>500</td>
<td>100 ea Turkey MRE</td>
<td>51-SW6</td>
</tr>
<tr>
<td>578</td>
<td>12 ea Desert boots (3AA)</td>
<td>45-Z43</td>
</tr>
<tr>
<td>300</td>
<td>5 ea Winter parka (XL)</td>
<td>41-PS3</td>
</tr>
</tbody>
</table>

Accept requests in same format as database

When you get a request, forward it to the commitment database

Volunteer 5: Warehouse East

Script

You are the database for a supply warehouse. You’re responsible for keeping track of what’s in the warehouse. You, however, are one of a set of several warehouses, and a centralized commitment database is used to keep track of requests. This allows coordination of trucks through several warehouses and simplifies re-ordering from the supplier. When you get a request, you tell the commitment database about it.

The what?

Notes

There are two significant differences between Warehouse West and Warehouse East. One is that the internal formats are different, including the stock numbers (for sanity, I just switched the two parts of the part number). The other is that this is part of a chain of warehouses and the commitment count is centralized.

First discussion: why are they different? Most reasons are variants on “they were developed independently, with different requirements.”

Second discussion: pros and cons of the centralized commitment data. This includes issues of consistency maintenance, as a given warehouse can now overcommit.
Commitment Database

Keep accurate list of what’s been promised from all the warehouses

<table>
<thead>
<tr>
<th>Count</th>
<th>Stock #</th>
<th>Description</th>
<th>Recipient</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>34-F78</td>
<td>Ping-pong table</td>
<td>Unit 4350</td>
</tr>
<tr>
<td>1</td>
<td>23-4G9</td>
<td>Humvee Transmission</td>
<td>Unit 2467</td>
</tr>
<tr>
<td>2000</td>
<td>51-SW6</td>
<td>Turkey MRE</td>
<td>Unit 6893</td>
</tr>
<tr>
<td>500</td>
<td>51-SW6</td>
<td>Turkey MRE</td>
<td>Unit 7491</td>
</tr>
<tr>
<td>1000</td>
<td>51-SW6</td>
<td>Turkey MRE</td>
<td>Unit 9215</td>
</tr>
<tr>
<td>81374</td>
<td>51-SW6</td>
<td>Turkey MRE</td>
<td>other units</td>
</tr>
<tr>
<td>78</td>
<td>45-Z43</td>
<td>12 ea Desert boots (3AA)</td>
<td>Unit 4350</td>
</tr>
<tr>
<td>32</td>
<td>98-W4K</td>
<td>8 ea Camo shirts (XXS)</td>
<td>Unit 6893</td>
</tr>
</tbody>
</table>

Accept commitments from warehouses
Accept requests in same format as database and assign to warehouses

Volunteer 6: Commitment Database

Script

You are responsible for keeping track of supplies that have been promised to units. This can happen in two ways: ideally, quartermasters will make their requests directly to you and you’ll assign the request to the warehouse closest to the quartermaster’s unit. Alternatively, a quartermaster may make a request of a warehouse and the warehouse will advise the commitment database.

OK, why are these databases organized differently? Well, yes, historical accident. But is one better than the other? What are the pros and cons?

Notes

The commitment database counts raw units. The reason is that the same items may be packed at various times with different numbers of units per box, pallet, or other warehouse unit.

This presents a marvelous opportunity for consistency errors. Try to get the group to give you a decision that this should count what’s committed for delivery, even if the number was raised from the quartermaster request to match warehousing units.

It’s fine to write policy requiring quartermasters to order centrally, but there needs to be at least an emergency option for direct orders. If the option is available, naturally the loophole will be used often.

Note that extensions to existing protocols are required for centralized ordering -- as specified, the commitment protocols only send information in one direction.
Volunteer 7: Warehouse East Communication Protocol

Script

You're responsible for accepting information from quartermasters, transmitting it reliably to Warehouse East, and reformatting it in the format Warehouse East expects.

Now, Quartermaster 4692, make your order.

<pause> Warehouse East, what are you going to do besides acknowledge? Yes, you have to tell the commitment data base. How?

Notes

Don't forget that you again need a decision about who converts quartermasters' unit counts to warehouse counts. Note that if the communication protocol does the conversion, the warehouse does not have exact counts to forward to the commitment database, even if it wanted to do so.

I set this up with separate communication protocols rather than a single versatile one. You could examine this question.
Warehouse East-Commitment Communication Protocol

When requests are submitted directly to Warehouse East, advises commitment database

Volunteer 8: WH East-Commitment Communication Protocol

Script

When requests are submitted directly to Warehouse East, you advise the commitment database. As with the other protocols, you'll have to convert between the formats of your two clients.

Notes
No new volunteer; return to Quartermaster 4692

Script

You’re an excellent quartermaster, looking out diligently for the needs of your troops. That’s why you remembered to order ping-pong balls as well as tables. «At this point, if QM 4692 didn’t do this, you’re making a joke and you should allow the oversight to be corrected.» How much do you trust the bureaucracy? Thanksgiving’s a really important holiday, isn’t it? You wouldn’t want anyone to do without a Turkey dinner just because the clowns over at the warehouse got busy with everyone else’s orders. Why not play it safe?

Quartermaster 4692, place order for Turkey MREs with Warehouse West, too. To do that, of course, you have to call upon Warehouse West Communication Protocol.

Do you notice anything about Warehouse West? Right, it isn’t advising the commitment database about commitments made. Better fix that, right?

Notes

It will cause the least confusion if you get the transaction completed in the old way before moving on to the next step and integrating this warehouse in the commitment system.
No new volunteer; return to Warehouse West

Script

Guess what? You've been integrated in the Grand Integrated Supply system. You must now tell the commitment database about everything you promise, when you promise it.

You'll have a major spasm while you get your current commitments recognized. After that, invoke the Warehouse West-Commitment Communication protocol whenever you process an order. Don't bother to amputate your internal accounting for commitments -- you'll break something if you modify the code, and besides, you may secede from the Grand Integrated System some day.

Notes

The current state of the Commitment Database actually includes the commitments that Warehouse West had made before this exercise started. It is therefore sufficient to record only the new order. Be sure to get the turkey dinners, and be sure to get the count right.
Warehouse West-Commitment Communication Protocol

When requests are submitted directly to Warehouse West, advises commitment database

Volunteer 9: WH West-Commitment Communication Protocol

Script

*When requests are submitted directly to Warehouse West, you advise the commitment database. As with the other protocols, you'll have to convert between the formats of your two clients.*

Notes
Volunteer 10: Commit Direct Protocol

Script

In the Grand Integrated Scheme of Things, policy requires that requests be sent to the commitment database rather than directly to the warehouses unless there's an emergency. Of course, given paperwork delays introduced by the extra processing step it's always an emergency, but let's make a request through the commitment database anyhow.

Notes

Note previous discussion about "emergencies" and the opportunity to introduce consistency problems.
No new volunteer; return to Quartermaster 2467

Script

Quartermaster 2467, don't you think you should be sure of your Thanksgiving dinners? If everyone else is double-ordering you may be left out in the cold. Order from the commitment database; for no particular reason we'll have it sent to Warehouse East via the Warehouse East-Commitment Communication Protocol.

Notes

QM 2467 will call on the Commit Direct Protocol, which will pass the order to the Commit Database, which will invoke the Warehouse East-Commitment Communication Protocol to communicate with Warehouse East.

Be sure that Warehouse East distinguishes between orders from quartermasters and orders from Commit; if it fails to do so, orders coming via this route will be double-committed.
Aggregate Database

Maintain overview of supply situation

For each stock #, determine number available
(add value from each warehouse and subtract
value from each commitment database)

Get values from warehouses with warehouse
protocols

Get values from commit with Commit Direct

Volunteer 11: Aggregate Database

Script

The People in Charge want to be able to figure out what’s going on. So they
want something called a Federated Database, in which all the information in all
the other databases is pulled together. In this particular system it’s called the
Aggregate Database.

It can get warehouse data from the warehouse protocols: Given a part number,
you can retrieve the count associated with it. Commit Direct can do the same
thing for the commit database. You combine these values to find various facts
about the Big Picture.

For example, you can compute the number of Humvee transmissions on hand by
getting Humvee numbers from the various databases. Remember that numbers
from warehouses enter with positive sign and numbers from the Commitment
Database enter with negative sign.

For demonstration purposes, we’ll generate each value needed whenever it’s
requested rather than trying to keep yet another version of the information
consistent with the rest.

Notes

Be careful here. Attentive students will automatically perform the unit counts<->
warehouse containers conversion. Stick with the rule that the protocol returns
the number in the count column. Yes, you’re going to get the wrong number.
That’s the point.

“Number on hand” means number in warehouses but not actually promised.
Don’t make too big a point of this, because on the next slide you’re going to
misuse the command.

This is a good place to discuss eager vs lazy evaluation of the derived aggregate
data.
The General

"How many Turkey Meals are On Hand?"

Volunteer 12: The General

Script

General, Sir! You are caring and sensitive, concerned for the welfare of the troops. The General wants to make sure that on the Most American of American Holidays, the troops have a genuine traditional turkey dinner. Or at least as close as you can come with MREs. So you ask "How many Turkey Meals are On Hand?"

Notes

The General asks the Aggregate Database, which asks each of the three protocols (Warehouse East Comm, Warehouse West Comm, Commit Direct) for the number in the count column.

The general has used "on hand" in a quite informal sense. We just gave it a precise definition that differs from the General's intuition.

At this point, databases should have the following values for turkey dinners:

- Warehouse East total of 500 cases in warehouse
- Warehouse West total of 40 cases in warehouse (some showing commitments)
- Commitment Database total of 89,874 units committed (89874 from original state, 5000 from 4 orders in this exercise)
The Surprise

Script

Aggregate Database should deliver the answer -89,284. At this point you lead a discussion of how that came about.

Notes

This slide is essentially the way the example came to me (I did fiddle with it a little, but the example is essentially real).

The error has three sources:

- Different units used for warehouse counts and individual items requested
- Unexpected behavior: duplicate orders by quartermaster
- Intuitive but incorrect use of technical command “on-hand”

The first of these has been discussed at length. The second have to do with human behavior; this provides an opportunity to discuss requirements analysis and user interfaces.

The point of doing this particular query is to have a climax for the exercise -- something to make the experience memorable, perhaps funny.

At this point you wrap up with a review of the highlights.