Object-oriented design: discovering objects with CRC cards. Reed Phillips, President of Knowledge Systems Corporation introduces the tape.

Question: How do I distribute the responsibility across collaborating objects so that together they solve my problem?

- Objects decompose problems. Responsibility must be correctly distributed among objects.
- Objects collaborate to solve problems. The tension between the individuality of objects and their interdependence makes designing object-oriented systems a challenge.

Ward Cunningham developed the CRC-card technique. He will demonstrate it with Kent Beck.

A CRC card looks like this:

<table>
<thead>
<tr>
<th>Name (Class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The problem is to design a defect-inspection system.

- Its purpose is to chops rolls of metal foil into sheets and to sort the sheets into two bins, according to a preselected product standard.
- Those that meet the standard go into one bin; those that fail go into the other.
- The system is equipped with a scanner, a chopper, and two airjets.

---

1These are notes from a tape by Reed Phillips, Ward Cunningham, and Kent Beck, used in this course by permission of the authors.
The scanner works by reading the amount of light that is reflected from the foil. The reflected light must be between certain values for the foil to be deemed good.

The chopper can be commanded to drop, cutting the foil into squares.

The foil may be chopped before it is scanned.

There are two airjets. One pushes foil to the left, one to the right. By custom, good foil is always thrown to the right.

Let’s consider the process from the point of view of the square.

The square doesn’t have physical existence before it is chopped. Perhaps we should identify potential squares.

There are also two airjets. There is some kind of dialogue that determines whether a square is to an airjet, and whether the airjet applies to it.

An airjet knows where it is located along the production line. It knows whether it is the “good” airjet or the “bad” one.
We are not yet sure if we want squares to be looking for airjets or airjets to be looking for squares.

---

**Airjet**
- Knows role and position.
- 
- 
- 

**Square**

There is also a chopper. Its responsibility is to change the state of the square.

---

**Chopper**
- Realizes squares.
- Operates physical device.
- 
- 

**Square**

There may also be a bin object that collects squares.

The airjet and chopper are similar, in that they know the state of a square and they change position. They will probably have a common superclass.

There is also a scanner.

---

**Scanner**
- Grades squares.
- 
- 
- 

**Square**
And the bin …

There are also other objects, like the production surface, the foil role, and the shaft encoder.

The production surface knows where the other objects are. It starts up, stops the line.

- It advances all the squares that are on the surface. Thus the square also has to have a position.

The square is passive; it just marches along under control of the surface.

At each click, the scanner asks, what square is in position for me to scan?

It also notifies devices when squares have arrived (so that, in the program, devices don’t have to check on each click whether a square is there).
Kent and Ward lay out the cards. A bin is very closely associated with an airjet. It is almost the same object.

Squares will be created good, and if it flunks, it stays flunked. System may keep track of how many of the squares are bad.

For now, time will march forward in units of shaft ticks.

Surface is saying, clock tick, clock tick, clock tick and a square arrived.

When a square gets to an airjet, the airjet asks whether it is a “good guy” or not. Then the surface needs to be told that the square is no longer on it.

Should the chopper precede the scanner?

- If the chopper comes first, it has to chop without knowing where the defects are, so you have to make uniform-size sheets.
- If the scanner comes first, you have the potential of creating different-size squares.

But then a square needs to know its size. And the surface has to work in smaller units of size than one square.
So the system keeps making the square longer and longer as long as it is bad. As soon as it finds a good place, it says, go back one, and that’s how long you are.

The scanner and the square jointly decide on how long the square is.

There has to be a maximum size for a square so that the airjet can handle it.

<table>
<thead>
<tr>
<th>Square</th>
<th>Surface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulates attributes.</td>
<td></td>
</tr>
<tr>
<td>Holds current position.</td>
<td></td>
</tr>
<tr>
<td>Stretches.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is the surface that creates the square.

<table>
<thead>
<tr>
<th>Surface</th>
<th>Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances squares.</td>
<td></td>
</tr>
<tr>
<td>Notifies devices of square arrival</td>
<td></td>
</tr>
<tr>
<td>Creates squares</td>
<td></td>
</tr>
</tbody>
</table>

We also have some passive objects.

<table>
<thead>
<tr>
<th>Production criterion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes required quality.</td>
<td></td>
</tr>
</tbody>
</table>
The production criterion is attached to a surface, which queries it in order to tell chopper when to chop. But every square consults it.

<table>
<thead>
<tr>
<th>Square</th>
<th>Surface</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Accumulates attributes.</td>
<td></td>
</tr>
<tr>
<td>• Holds current position.</td>
<td>Square</td>
</tr>
<tr>
<td>• Stretches.</td>
<td>Production</td>
</tr>
<tr>
<td>•</td>
<td>criterion</td>
</tr>
</tbody>
</table>

Every square that is created by the surface is tagged with the production criterion that was used to create it.

So the production criteria can be changed in the middle of a run.

Now let’s review incidents that show how CRC cards helped to find the objects in the application.

• Create objects only as they were needed.
• Invest effort in phrasing the responsibility.
• Determine appropriate collaborations.
  The collaboration relationship is not necessarily symmetric.
• Use spatial layout to show how the application fits together.
• Stack cards to simplify design.
  Stacking becomes a form of abstraction.
• Develop scenarios to illustrate what can happen during operation of the system.
• Reallocate responsibility to remove flaws in the design.
• Using cards help you role-play by acting out the behavior of the object.