Rules, Processes, Procedures, and Logic

Review -- Types of Components in Systems in the UML

Introduction -- Business Rules and Procedures

Example -- History of how banks clear checks

Campbell-Kelly10b

Needed Skills for analyzing procedures and processes

Example -- Department Honors

Example -- A Tax Form

Example -- DiscountsRU

Introduction to Capturing Rules and Procedures

Principle -- First determine the variables in the rule.

Principle -- Use numerical variables whenever possible.

Principle -- Classify Rules as REQUIRED or NATural

Notations and Tools for Capturing Rules and Procedures

Text

Diagrams

Pseudocode and Structured English

Symbolic Logic and Mathematical Formula

Story -- Do the Math

Mathematical Logic for modelling Business Rules

Using Discrete Mathematics to express Data Base Operations

Tables

Prototypes

Hint -- Reify Complex Rules -- encode them as data

Hint -- Use tables to sort out rules before you write code

Frequently Asked Questions on Detailed Models

What tools should I use to draw diagrams?

Is there professional standard software package for producing diagrams

What is the cheapest graphic tool

What is the cheapest CASE tool

Is advanced modeling terribly difficult

How does a DFD differ from an activity diagram?

How do you show how control flow is implemented in a DFD

Who designed activity diagrams and when where they designed

Is there a fork and join in old fashioned flowcharts

Online Resources and Exercises

User Stories on the Wikipedia

Scenarios on the Wikipedia

Decision Trees and diagrams on the Wikipedia

Big Example Activity Diagram with Swim Lanes

Advice on Activity Diagrams Online

Object Management Group on the web

Flowcharts

Tutorial on ANSI Flowcharts

Buying an ASME template

State Charts on the web

Pseudocode on the Wikipedia

Writing Mathematics on a computer

Venn Diagrams

Tables on Wikipedia

More on tables

Review Questions and exercises

Abbreviations

Links
Review -- Types of Components in Systems in the UML

This set of notes focuses on the Rules and Procedures that make a system work...

Algorithms in the News

An algorithm is a detailed procedure -- stated in enough detail that it can be executed on a computer. Here is a rare event -- an algorithm in the news [story.php?storyId=114259241](story.php?storyId=114259241) (How Do You Find A Job? Ask The Algorithm).

However you can expect more news items on algorithms.

Welcome to the algoworld

2. Algorithms are an invisible part of the 21st century and carry a spurious authority.
3. Gives examples of algorithms doing ridiculous things.

Introduction -- Business Rules and Procedures

All enterprises have their own way of doing things. These are often very well defined. In many cases they are complex legal requirements. Your new software etc. must fit with the rules and implement the procedures. One problem that a software developer or development team must solve is being able to capture rules and procedures. Luckily there are many tools you can use. The best of these are covered in these notes. The same tools/techniques can be used to plan what parts of a system will have to do -- specify the requirements for hardware and software.

Notice that a detailed model can include assumptions about the technology used to implement the model: this is a Physical Model. It can also be independent of the technology and so have a longer useful life -- these are called Logical Models. In these notes mentally classify each technique or tool as best for either logical or physical models.

Example -- History of how banks clear checks

Here is a brief article describing some complex manual procedures that solve an important problem. Notice how the physical systems evolve but still meet the logical requirements:

Campbell-Kelly10b

1. Martin Campbell-Kelly
2. Historical Reflections: Victorian Data Processing
4. =HISTORY DATA PROCESSING LOGICAL vs PHYSICAL BABBAGE CLEARING CHECKS USA CHEQUES UK
5. Shows an efficient solution to a problem in the form of a multi-agent procedure.
6. Traces the evolution of Bank Clearing Houses in the UK (as documented by Babbage) and then in the USA.
7. Argues that similar algorithms are still used in modern information technology.

Needed Skills for analyzing procedures and processes
There are four skills that you need whatever techniques you use. (1) Discovering the variables (if any). (2) Defining the possible scope for the rules and writing rules/procedures that fit these possibilities. (3) Covering every possibility -- this is called completeness. (4) Giving only one response for any possibility -- this is known as consistency. (5) Getting the rules right -- this is correctness.

You will find that writing correct, complete and consistent rules/procedures to be harder than you might think.

**Example -- Department Honors**

What do you make of this description: [honors.html], is it consistent and complete?

**Example -- A Tax Form**

Here is a complex manual algorithm [f1040sa.pdf] how do you translate it into a more manageable form ready to program? How do you know that the program is correct?

**Example -- DiscountsRUss**

A certain department store had a complicated scheme for calculating discounts. Here are the rules:

```plaintext
RULES ::= following
Net
1. (R1): If a person has a club card and today is a Tuesday then the discount is 10%.
2. (R2): If a person has a club card and today is not a Tuesday then the discount is 5%.
3. (R3): If a person is a senior and has a club card then the discount is 10%.
4. (R4): If a person is a senior and has not got a club card then the discount is 5%.
5. (R5): If a person is not a senior and has not got a club card then the discount is 0%.
(End of Net RULES)
```

I'm glad that I don't have to program these discounts into the Point Of Sale terminals! The RULES above have a problem -- they are inconsistent: sometimes they give two discounts to a person (exercise: when?). It is also difficult to find out if the rules are complete -- is every possible case covered?

**Introduction to Capturing Rules and Procedures**

To express complex rules and conditions we need precise notations: decision tables, and/or tables, logic trees, Karnaugh maps, activity diagrams/flow charts, mathematical logic, ... plus other techniques discussed in [../cs556/]

Formal Methods. Some of the best known are discussed below. Formalizing and checking procedures and rules is time consuming but essential. Once found, there are many ways of resolving inconsistencies and completing the rules. However these should not be invented by a programmer or an analyst -- you need to talk to the people involved in the system -- the stakeholders -- to resolve inconsistent and incomplete rules and procedures.

**Principle -- First determine the variables in the rule.**

All rules refer to things that vary. Discovering them is important. Defining their possible values is vital. When they are Boolean (either true or false) they are called conditions.

For example, in the RULES above we have six obvious Boolean variables.

1. the person has a club card.
2. today is Tuesday.
3. the person is a senior.
4. the discount is 10%.
5. the discount is 5%.
6. the discount is 0%.

Another formulation has three Boolean variables and a numerical discount

1. the person has a club card.
2. today is Tuesday.
3. the person is a senior.
4. the discount is either 0%, 5% or 10%.

Giving them short names is a helpful shorthand.

It also helps to document that part the variables play in the rule you are working on: some variables are given and others are goals. The givens are also known as inputs and the goals as outputs. For example
2 **GIVENS** := following
   1. the person has a club card.
   2. today is Tuesday.
   3. the person is a senior.

3 **GOAL** := following
   1. the discount is either 0%, 5% or 10%.

The rule will either show you how to work out the goals from the givens or it, at least tells how the goal variable is constrained by the given variables.

**Principle -- Use numerical variables whenever possible.**

Some notations (below) force you to express every variable as a Boolean variable (values True or False). The better notations allow a variable to have other values. Numbers are very useful for many rules. The can represent money, discounts, heights, counts, widths, velocities, and so on.

**Principle -- Classify Rules as REQuired or NATural**

Rules that appear in systems -- also known as *constraints* can be -- can be classified as being *natural* because they are true automatically. Other rules are *required* -- they must be enforced by the system we are designing. These are abbreviated NAT and REQ. The system itself will enforce certain rules. Call these SYS, and then we must have

\[ \text{if NAT and SYS then REQ, for the system to meet the requirements.} \]

**Notations and Tools for Capturing Rules and Procedures**

Choose the notation and media that fits your needs now and in the future. This means being prepared with several techniques/tools. Here is quick list of the techniques covered below.

1. **Text**
   1. User Stories
   2. Scenarios

2. **Diagrams**
   1. Plans
   2. Activity Diagrams
   3. Earlier Flowchart-like notations

3. **Pseudocode and Structured English**

4. **Symbolic Logic and Mathematical Formula**

5. **Tables**
   1. Truth Tables
   2. Karnaugh Maps
   3. Function Tables
   4. Decision Tables
   5. And/Or Tables
   6. Extended Entry Tables

6. **Tree Diagrams**

7. **Prototypes**

**Text**

Text is natural. But it is hard to read and difficult to write clearly. Natural language is also ambiguous. There are few tools for analyzing text data. It is almost impossible to read a complex set of rules or a description of a complex procedure and spot errors and inefficiencies in it. There are horror stories of teams faced with a thousand pages of densely written rules and procedures that have to be embedded in their software. On the other hand, you should include text in requirements documentation to explain the thinking that goes into the more formal definitions, tables, diagrams, and formulas.

**User Stories**

Writing a one paragraph description of a feature or property that the user has asked for is a a popular and agile requirements technique. A "user story" is a short, one paragraph description, of something that the user wants. If it can not fit on a 3×5 card then it is too long. If it has lots of technical details it is not a "user" story. User stories are simple enough not to need much coverage here. You can get details from the Wikipedia, link below.

**Scenarios**

A *scenario* is another simple text based technique: Scenarios are a numbered list of actions by various actors. They
Plans

When you have a set of activities that must be completed but you have no required order then you can use Network Planning and the Critical Path Method to document, plan, manage, and control them.

Network plans are essentially special Activity Diagrams (below).

Tree Diagrams

Sometimes you have a complex decision to capture in your requirements. Tree diagrams are a popular way of mapping out a complex sequence of conditions. You can use an activity diagram or a notation like this

Decision Trees

If you want to learn more you can use the Wikipedia. I don't require you to use these in quizzes, finals, or exercises. You may find them helpful in presenting your project. You may choose to use them in exercises and tests if you think they will help. Tree Diagrams are essentially specialized Activity Diagrams (next).

Activity Diagrams

Activity diagrams are one of the thirteen UML diagrams. They are an excellent way to record existing procedures, new procedures, programs, and processes. They can show both complex logic and complex interactions between activities in a process.

Here is an example of these symbols in use describing a procedure for handling Customer Orders.

Here is the same example procedure with swim lanes that allocate the activities to different parts of the system or to different actors.
There are a lot more features available to you in the standard, for example here is a link to an example of a data store.

Notice the <<data store>> in the above activity diagram. This indicates the systems database. The <<select>> stereotype indicates the kind of access that should used.

**Flowcharts -- Activity Diagrams before the UML**

Here is an image of the symbols we used back in the 1970s for flowcharts.

The Dia and Visio graphic programs provide ready-made palettes of these symbols when you need to produce a final/pretty diagram. Or when you know the procedure is going to change and so want an editable format.

The simplest set of symbols is the ASME standard set. It has precisely five symbols: operation, delay, inspect/test, store/retrieve, and transport. An efficient process minimizes everything but operations!
An important rule for flowcharts and activity diagrams

They always have a special START node to get the activity/process started. They usually have a number of STOP nodes to terminate the activity.

A Flow Chart is not a DFD

Do not confuse these. A flowchart shows the sequences of activities. A DFD shows parts of a system and how they are connected. A flow chart can have decisions and branches but a DFD does not. A flowchart has a start and a finish, a DFD is running all the time.

Use flow charts and activity diagrams to document the detailed behavior of processes inside a DFD -- if the process is complicated and interesting.

State Charts and Protocol Machines

Sometimes a business will have rules that determine the order in which events must occur. This is subtly different to describing the sequences of activities. Here we want to be sure that someone does not draw a pension before they retire or after they are dead, for example. The natural way of handling these rules is to define states (nodes in the diagram) and the events that change states (arrows with [conditions] and event names on them).

The UML provides a type of diagram -- a State Machine or State chart. It shares a lot of features with an activity diagram. The key difference is the appearance of events on the transitions. Here is an example from the draft UML2.0 standard.

These state machines are studied in great detail in Computer Science theory. Surprisingly they are (1) useful and (2) supported by some libraries, for example [http://www.skorks.com/2011/09/why-developers-never-use-state-]
Decisions in Diagrams

Notice that complex decisions sometimes appear in flow charts and activity diagrams. In activity diagrams the conditions are written in square brackets on the transitions: \([x>5]\), \([x<5]\), \([x=5]\). In flowcharts the condition is written inside the diamond and the transitions are typically labeled "yes" and "No" or "True" and "False". Sometimes the conditions can be difficult to document. Give them a meaningful name (\([\text{no discount}]\)) and use the other techniques described here (Boolean expressions like \([\text{Senior} \& \& \text{Tuesday}]\) to document the details.

Pseudocode and Structured English

In the 1960s Computer Scientists proved that we only need three structures to express any algorithm. They are: **Sequence, Selection, and Iteration:**

![Diagram of Sequence, Selection, and Iteration]

Programmers have since learned to use them and to express required algorithms and program designs using a structured version of English. Pseudocode (*sue-do-...* not *suede-oh...*) is an imitation structured programming language. It typically mixes a natural language and structures like IF-THEN-ELSE-END IF and WHILE-DO-END WHILE.

Here is a sample of traditional Pseudocode:

```plaintext
IF person has club card THEN 
  IF today = Tuesday THEN 
    discount := 10% 
  ELSE 
    discount := 5% 
  END IF 
ELSE 
  discount := 0% 
END IF 
ELSE discount := 0% 
END IF 
```

The use of "=\(\)" is not like it is in C/C++/Java/PHP/... but more like that writing conditions the 1960s. In the above "=\(\)" indicates that a new value is assigned to the variable on the left hand side. The "==\(\)" is a test for equality. Using "===\(\)" is an error.

Notice that every "IF" has a single matching "END IF". An IF may or may not have an "ELSE".

Here is an example with a WHILE loop.

(Algorithm F):

```plaintext
factorial := 1 
WHILE n > 1 DO 
  factorial := factorial \(\times\) n 
  n := n - 1 
END WHILE 
```

Every "WHILE" has a matched "END WHILE"

Notice that you can write and process Pseudocode with your favorite programming editors. What makes it "Pseudo" is that you can not compile or interpret it. On the other hand it has equally strict rules:

1. Use capital letters to show structure.
2. Every IF has a THEN.
3. Every IF has an END IF.
Every WHILE has a DO.
Every WHILE has an END WHILE.

Here is the BNF syntax for

```
Net
  condition := given.
  variable := given.
  expression := given.
  assignment := variable "=" expression.
  sequence := statement # statement.
  statement := line | loop | selection.
  loop := "WHILE" condition "DO" eol sequence "END WHILE" eol.
  selection := "IF" condition "THEN" eol sequence O("ELSE" eol sequence) "END IF" eol.
  line := (assignment | other action) eol.
  eol := end of line.
```

Not following this syntax in this course is an error.

Symbolic Logic and Mathematical Formula

Some rules are best expressed using traditional mathematics.

For example, in a game, a missile will follow a trajectory described by

```
Net
  v:Real = given, vertical velocity.
  g:Real = given, acceleration due to gravity.
  y0:Real = given, initial height.
  x0:Real = given, initial horizontal position.
  t:Real = given, time.
  x:Real = goal, horizontal position at time t.
  y:Real = goal, height at time t.
  y = y0 + v * t - g t^2 / 2.
  x = x0 + u * t.
```

The formulas are precise and short. They may not be stakeholder friendly. But some stakeholders, with a scientific background will expect you to use mathematics. Setting out the variables is helpful and takes most of the time.

Story -- Do the Math

In this story a project was finished quickly by expressing the user's problem in mathematics and looking up the solution in a book.

In "Imperial Chemical Industries" (UK) in the 1960's the Sodium Production group asked Computer Services to find the optimal rate of flow of mercury through their sodium electrolysis bath. A young co-op student (me) was tasked to work with them, develop a model, solve it, use the mainframe to implement the solution, and provide results that let them choose the best flow. I talked with the people and developed a special kind of model: (a Partial Differential Equation), looked up the formula solving the equation in a text book, translated the formula into code, and tested it (worked second time). This took about less than 3 days (one test per day). But I had a whole week before I met the clients again so I modified it to give graphic output.

When I showed it to the user he: (1) gave me the correct parameters for the model, (2) rejected the results with four decimals of accuracy -- he could only measure to 1 place, and (3) said the graphics were more help. I removed the numbers.... and the result was a program that solved their problem.

Summary: by using mathematics and three iterations the whole project was finished in 2 weeks.

Mathematical Logic for modelling Business Rules

In business most rules have conditions have "If .... then ...." in them and so you need a way to express conditionals. Symbolic or mathematical logic can help.
So the example **RULES** (above) we might be expressed in logic like this. (the '@' indicates a true/false (Boolean) proposition)

Net

1. \( p@\) = given, the person has a club card.
2. \( q@\) = given, today is Tuesday.
3. \( r@\) = given, the person is a senior.
4. \( s@\) = goal, the discount is 10%.
5. \( t@\) = goal, the discount is 5%.
6. \( u@\) = goal, the discount is 0%.

And then the requirements might be written something like this

\((r1)\): \( p \land q \Rightarrow s\)
\((r2)\): \( p \land q' \Rightarrow t\)
\((r3)\): \( r \land p \Rightarrow s\)
\((r4)\): \( r \land p' \Rightarrow t\)
\((r5)\): \( r' \land p' \Rightarrow u\)

(End of Net)

Notice that this approach does not help the typical stakeholder. More, it does not help the typical programmer either. There have been experiments in using logic and discrete math to describe the behavior of a system in abstract form and then to analyze the model to see if the ideas work or to search out unwanted behaviors. We have tools that test logical statements for completeness and consistency. These are a part of [../cs536/](file:///u/faculty/dick/cs536/) Formal Methods. They have been used successfully to debug real projects.

**Using Discrete Mathematics to express Data Base Operations**

The theory of data bases is based on Discrete Mathematics. The boxes in a normalized ERD are just sets, and the lines are functions: many-to-one relations or mappings. Each table in a relational data base is just a set of tuples. The data base operations add and delete rows/tuples or change items in certain rows. You can use the operations in set theory -- union, intersection and complement to express some operations. Other operations need either specialized symbols of "set comprehensions".

For example (Taken form "Software Development with Z" by J B Wordsworth 1992) suppose we are keeping track of students who enrol in a class, take a test, and may be awarded a certificate. Here I use my MATHS notation for set operations.

6. **Class_managers_assistant** := following,

Net

1. Students := Sets = given.
2. Enrolled := @Student, @ indicates a subset.
4. |- |Enrolled| <= Maximum_enrolled.
5. Tested := @Student.
6. Passed := @Tested.
7. Failed := @Tested.
8. |- Tested = Passed | Failed. Set theory 'union' is shown above. "Tested students are either Passed or Failed'.
9. |- No (Passed & Failed). Set theory intersection is shown above. "Nobody is both Passed and Failed'.
10. |- No(Enrolled & Tested).

(End of Net)

7. Initially := **Class_managers_assistant** with \( \{ \text{Enrolled} = \text{Tested} = \{} \} \).

For \( s : \text{Students} \), Enrol\( (s) := \text{Class_managers_assistant with} \{ \text{s not in Enrolled|Tested. Enrolled'}=\text{Enrolled}\{s\}\} \).

The prime (') indicates the next value of variable.

9. For \( s : \text{Students} \), \( r : \{\text{pass, fail}\} \), Test\( (s, r) := \text{Class_managers_assistant with following,} \)

Net

1. |- s in Enrolled ~ Tested, Set theory complement but not.
2. |- Enrolled' = Enrolled ~ \( s \), remove s from enrolled students.
3. |- If \( r=\text{pass then Passed}=\text{Passed} \{\text{s}\} \).
4. |- If \( r=\text{fail then Failed}=\text{Failed} \{\text{s}\} \).
5. (above)|- s in Tested'.

(End of Net)
Using discrete mathematics clarifies our thinking. It gives a very rigorous definition of how the database can evolve. It allows us to prove useful properties and helps us get programs that are bug free. There are even tools that let you test whether a set of requirements for a system achieve desired results. However, this is not a popular technique in industry. If you ever need to, you can find out more at [../maths/] or search the web for the language Z ("Zed").

**Tables**

Tables are an excellent way of analyzing and expressing complex logical rules. I've used them for 30+ years. They have the precision of symbolic logic but they are also client friendly. This section of notes shows how the Discounts'R'Us RULES can be expressed using several tabular notations. Some of these notations uncover the problems with the rules... which?

**Truth Tables**

These have one column for each variable. Each row lists the values of the variables: T=true, F=false. The following table shows the possibilities for a discount of 10%.

<table>
<thead>
<tr>
<th>Club card</th>
<th>today=Tuesday</th>
<th>Discount=10%</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

(Close Table)

Each column that you add doubles the number of rows. So if we documented all the given RULES we would need 32 rows. This is good for symbolic logic but clumsy and verbose used for software requirements. The table does not help discover inconsistencies and incompleteness in a set of conditions.

**Karnaugh Maps**

By using two dimensions, we can save space and compress all 8 cases into a "matrix" of possibilities. For each case we put the result(T/F) in it's cell:

<table>
<thead>
<tr>
<th>Club card</th>
<th>today=Tuesday</th>
<th>Discount=10%</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

(Close Table)

But this is still clumsy and unhelpful. The full example has 5 variables and 32 cells. Karnaugh maps with more than 4 T/F variables are complex.

**Function Tables**

A famous computer scientist and software engineer, David Lorge Parnas, in the 1990s, made the next step: extend the table to allow non-truth-values in the cell. These tables are used to describe functions: rules that map conditions into results. We make one cell for each combination of the inputs and put the result in the cell. This makes it easy to check the requirement for completeness (no blank cells), and consistency (one value per cell).

For example

<table>
<thead>
<tr>
<th>percent_discount</th>
<th>:=following.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Club Card</td>
<td>Age</td>
</tr>
<tr>
<td>Club Card</td>
<td>Senior</td>
</tr>
<tr>
<td>Club Card</td>
<td>not Senior</td>
</tr>
<tr>
<td>no Club Card</td>
<td>not Senior</td>
</tr>
<tr>
<td>no Club Card</td>
<td>Senior</td>
</tr>
</tbody>
</table>
Notice that recording all the rules in this kind of table instantly identifies the problem with **RULES**. Key point -- a function table exposes an inconsistency. It will also show up incompleteness. It lets us discover and display to the stakeholders, the problem with their requirements. It is a good tool for improving them.

**Decision Tables**

These date back to the 1960's and are a *semi-formal notation*. This means that they have a well-defined mathematical base but they were used before the theory was invented. Each decision table has two parts. The top half lists the conditions and values like a truth table -- but many use Y=Yes, and N=No instead of "T" and "F". Conditions are also be shown as "-" which means "don't care". The bottom half indicates the actions. An action to be executed is marked with an "X". An extra row counts how many cases are covered in each column. Each don't care doubles the number of cases in that column. These numbers are used to check the table for completeness and consistency:

<table>
<thead>
<tr>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Condition</strong></td>
</tr>
<tr>
<td>club card</td>
</tr>
<tr>
<td>Tuesday</td>
</tr>
<tr>
<td>Senior</td>
</tr>
<tr>
<td>Action</td>
</tr>
<tr>
<td>discount=10%</td>
</tr>
<tr>
<td>discount=5%</td>
</tr>
<tr>
<td>discount=0%</td>
</tr>
<tr>
<td>Count</td>
</tr>
</tbody>
</table>

Each column describes a condition and the actions that are required. In general, several actions can be done (several X's in a column) for a set of conditions. Indeed you use "A", "B",... to indicate the sequence of actions to be executed.

Notice that each column covers 2 cases and so there are 10 cases covered in the table, even tho' there are only 8 possible combinations of T and F! This simple check should make us search for the inconsistency in the original requirements. Again a table proves helpful in spotting incomplete or inconsistent rules.

The main advantage is that users and managers have no problem understanding them. The main problem with these tables was maintaining them in parallel to source code. To fix this problem people developed decision table compilers. These generate source code from decision tables (one written in COBOL by a BS student of mine in the 1970's).

**And/Or Tables**

A piece of research (the Traffic Collision Avoidance System -- TCAS II in 1995... at UC Irvine ) rediscovered the value of tables. They used "And/Or" tables to document the conditions for states to change. Their clients liked them. The key idea is that each column is "and-ed" together, the columns are "or-ed". The researchers developed tools to check And/Or Tables for completeness and consistency.

10% discount OR
A Club card | T | N |
D Tuesday | T |  |
N Senior | F |  |
5% discount OR
A Club card | T | N |
D Tuesday | T |  |
N Senior | F |  |
0% discount OR
A Club card | T | N |
D Tuesday | F |  |
N Senior | F |  |

A graduate student at CSUSB has developed web-based tool to generate AND/OR tables from formulas. I have been working on algorithms that can carry out Boolean operations on And/Or tables. A BA student has developed an initial C++ API framework for manipulating And/Or tables.

**Extended Entry Tables**

Notice that all types of tables that only use "T" and "F" can be extended to tables that use other values for variables. This reduces the size of the table but makes them harder to check.
Prototypes

One way of exploring complicated requirements is to program them in a throw-away (Bread board) prototype. Review [a6.html#Prototypes] to see what I mean.

Notice that prototypes are OK for checking out how obvious requirements work. They can not help with checking for obscure conditions that are inconsistent or cases that are not defined. The tables and other techniques in this set of notes can do this.

**Hint -- Reify Complex Rules -- encode them as data**

When you have a complex piece of logic to express it may pay to encode it as data. This typically simplifies the algorithm. If the data is held on a data base then it can slow down the program. But when the rules change (and most rules in real life change) all you have to do is change the data. This can even be set up as a process done by a stakeholder. This simplifies maintenance and reduces the total cost of ownership.

Here is an example: the rules of Tic-Tac-Toe are simple to state in English. Consider just the rules that determine if \( x \) has won: There must be a row with 3 \( x \)s, a column with 3 \( x \)s or a diagonal with 3 \( x \)s. It is easy to encode the board as an array of 9 characters. Writing out the rule for winning in a table or in symbolic logic takes time (exercises below).

Diagrams, pseudocode, and coding are worse. One technique is to list the the 8 winning configurations (3 rows, 3 cols, 2 diagonals) in a constant array and just run through them in a simple loop. The program is much simpler because part of the logic is now data.

Here is an example where putting part of the program in a data base makes sense. The fare on our local bus system depends on the type of pass (one trip, one day, ...) and the type of person (senior, disabled, student, other). More ... the values change every year. It would be a mistake to code this in a program:

```
IF person is senior AND pass is 31-day
    THEN cost is 23.50
...
```

Instead add entities to the data base and look up the value. Add a simple process that lets the right person in the bus company change the values. You can now use the same data to prepare flyers, fare sheets, and time tables....

**Hint -- Use tables to sort out rules before you write code**

In my experience complex logic is nearly always best analyzed using one or more tables. This will tell you if any cases are missing, or if the logic is inconsistent. After you have one this and resolved the problems it is worth going to pseudocode and prototypes.
I'm not sure but there is a good open source product.... Umbrello tends to crash. Argo is at [http://argouml.tigris.org/](http://argouml.tigris.org/) but does not support standard UML2.

**Is advanced modeling terribly difficult**

By the end of CSci375 you'll find modeling a simple process. The catch is that getting a useful and correct model can take time and thought. Modeling is a disciplined way of thinking. Of course, any kind of thinking tends to be hard work.

**How does a DFD differ from an activity diagram?**

A DFD is made of stores and processes. An activity diagram has activities, objects, and decisions. Activities do something and then cease to exist. When an activity outputs an object it outputs just one. It acts as a signal to a new activity to take over and absorb the object. The processes in a DFD always exist, they just pause when there is no data to process. In a DFD the data flows are buffered. Thinking electricity in a wire or water in a pipe. They show a sequence of objects moving from a process or store that does not go away -- the parts continue to exist and continue to work until the system is destroyed. A DFD shows the processes and data that exist and has no starting point or stopping point. The parts of the system: stores, processes, flows, and entities are assumed to neither stop or start, however they may rest for a while when nothing has to be done. They are like parts of a mechanical clock.

As a rule an activity diagram shows a lot more detail than a DFD. One process in a DFD may take a whole activity diagram to describe in detail. An activity diagram and a flowchart always has a START node. And nearly all of them have a STOP node. DFDs don't have these nodes.

**How do you show how control flow is implemented in a DFD**

You don't! If control flow is needed use an activity diagram.

What you can do in a DFD is show a process with a flow of data entering it and two flows leaving, and each item in the flow is sent out on one or other of the outputs. Here are two example DFD processes.

Here is what happens when you draw a DFD of a process that can produce different sets of outputs depending on the values that are input:
Who designed activity diagrams and where they designed

The group working for the Object Management Group (OMG) developed the specifications from about 2000 to 2004. It is an industry sponsored standard based on unifying several previous ways of showing complex processes and protocols.

Is there a fork and join in old fashioned flowcharts

Yes. They are just like the ones in UML Activity diagrams. Fork and Join date back to the 1970s.

Online Resources and Exercises

User Stories on the Wikipedia
[User_story]

Scenarios on the Wikipedia
[Scenario]

Decision Trees and diagrams on the Wikipedia

The Wikipedia article [Decision_trees] is about the use of decision trees in artificial intelligence and data mining. However [OBDD] Describes a version of tree diagrams that have been extensively analyzed by computer scientists called "Ordered Binary Decision Diagrams". We don't have time to chase OBDDs any further in this class. I often cover them in my formal methods ../cs556/ class.

Big Example Activity Diagram with Swim Lanes

It shows an Activity Diagram with Swim Lanes showing how faculty and students interact over assigned work. Please print out this (large) graphic: [07ActivitySwimLaneEtc.pdf] as part of this set of notes.

Advice on Activity Diagrams Online

When you want to know more about activity diagrams for a project, you used to be able to find the "UML2.0 in a Nutshell" and the UML2.0 reference manual online and free. But no longer:-(

Try these [activityDiagram.htm], [activityDiagram.htm], [31863#activity-diagrams], and [activity.htm] instead

Object Management Group on the web

For hardcore descriptions of the UML and other object oriented things:
OMG := See http://www.omg.org/

Flowcharts
The earlier notations where ANSI/ECMA/IBM standard Flowcharting symbols. Here is a template or stencil for the symbols that you can get from [http://www.draftingsupplies.com/](http://www.draftingsupplies.com/) (if you ever need to!).

**Tutorial on ANSI Flowcharts**

If you want to learn how to do ANSI (American National Standard) flowcharts then start by studying the PDF found here [citation.cfm?id=356566.356570](citation.cfm?id=356566.356570) (you'll need to join ACM or be on campus...) -- and it killed my Firefox when I tested it today.

**Buying an ASME template**

[1218i.jpg](1218i.jpg)

**State Charts on the web**

If you want to know more about state charts check out the 3rd edition of Fowler's "UML distilled" or [stateMachineDiagram.htm](stateMachineDiagram.htm) (Ambler's Agile Modeling Site).

**Pseudocode on the Wikipedia**

The Wikipedia [Pseudocode](Pseudocode) entry is an excellent resource.

**Writing Mathematics on a computer**

For my ideas on expressing symbolic logic and mathematics on computers see [../maths/intro_logic.html](../maths/intro_logic.html) and [../maths/intro_characters.html](../maths/intro_characters.html)

Traditional mathematical notation needs a tool like Donald Knuth's more complex ΤΕΧ language. See my notes [../samples/languages.html#TeX](../samples/languages.html#TeX) or the "Equation Editor" found in most word processors from [http://www.mathtype.com/](http://www.mathtype.com/) Also see [TeX](TeX) on the Wikipedia.

**Venn Diagrams**

Venn diagrams are an excellent presentation tool for simple logic with no more than three (3) sets or predicates. However they are no help in more complex situations. Interesting examples of Venn Diagrams, graphs, and diagrams can be found at [watch?v=lWWKBY7qX_0](watch?v=lWWKBY7qX_0) and [http://thisisindexed.com/about/](http://thisisindexed.com/about/) (blog). Note -- I do not endorse the content in these web sites. They just show some informal ways of using mathematics.

**Tables on Wikipedia**

The Wikipedia has notes on [Truth_table](Truth_table) and [Decision_table](Decision_table) if you need to find out more.

**More on tables**

Look at [notn_9_Tables.html](notn_9_Tables.html) for more on tables.

---

**Review Questions and exercises**

1. Use Google to search for recent news items about "algorithms".
2. Check the formulas, diagrams, and tables in this handout for mismatches with the Discounts'RUs requirements.
3. Put these steps in analyzing or documenting rules and procedures into the best order: prototype, pseudocode, tables, text, variable (in alphabetical order)
4. List the 15 or so techniques in this set of notes (from memory). Which are best for physical models and which for logical models?
5. Here TBA is a set of requirements expressed informally. Use any techniques that you know do discover any incompleteness and/or inconsistencies in these requirements.
6. Here are two sets of requirements concerned with the encoding of Universal Resource Locators in the HTML. Do they always describe the same encoding? If they don't give an example and how it is different.
   
   **URL_encoding :=**

   Each character in the string is encoded as follows: If it is a letter or a digit it is copied, if it is a space then it is replaced by a plus sign. Otherwise the character is replaced by a percent sign followed by the hexadecimal code for the ASCII code of the character.
IE4_URL_encoding :=
Net
Each character in the string is encoded as follows: If it is a letter or a digit it is copied, if it is a space then it is replaced by a plus sign. Plus signs are copied as is. Otherwise the character is replaced by a percent sign followed by the hexadecimal code for the ASCII code of the character.

Find a set of complicated requirements like the Discounts’RUs example and attempt to use any of the techniques in this section of notes to document it. Can you discover any inconsistencies or any incompleteness?

Here [hailstone.png] is an activity diagram describing the Hailstone sequence. Work out a scenario that shows all the successive values of \( n \) if the value input is 17.

Draw a table showing one step of the Hailstone sequence.

Write out some Pseudocode for the Hailstone sequence (above).

Draw an activity diagram for calculating the factorial of \( n \) from the Pseudocode Algorithm_F in the page above.

Here [activitydiagram.gif] is an activity diagram. Translate it into English!

What is missing from this State Machine: Hint -- this correction [http://xkcd.com/851_make_it_better/] has the same errors.

Consider the two example DFDs that split up data flows (above). Draw an activity diagram for separating even and odd numbers. Write Pseudocode for separating undergraduates from graduates.

What do you use an activity diagram or flowchart for?

How does a DFD differ from an activity diagram -- list all the ways.

How does an ERD differ from an activity diagram -- list all the ways.

What is a scenario? How does it relate to a user story and to an activity diagram.

Describe Pseudocode.

Draw a flowchart/activity diagram that models cost benefit analysis.

What is a tree diagram? Include an example in your definition.

Draw a tree diagram that determines if a year (A.D.) is a leap year or not.

What do we call a program that is not useful but demonstrates some of the requirements on a system? How does this differ from an iteration?

Draw a flowchart/activity diagram that models cost benefit analysis.

Explain and define an algorithm for solving the quadratic equation \( ax^2 + bx + c = 0 \) for all possible cases with real a, b, and c. See the DFD [SolveQuadDFD.png].
End