Recall from previous lecture…

Understanding customer requirements is critical to success of any software development effort

Major process tasks:

– Requirements *elicitation/analysis*

– Drafting a *requirements specification*

**Review:** What are the (meta-) requirements on a requirements specification?
Software specification

(1) How *NOT* to write a specification

Topics:

– Problems with natural language specifications
– Case study in functional specification: Text formatting
– Companion paper:
Specification

**Defn:** Statement of an agreement between producer (of a product or service) and consumer (of that product or service) or between an implementer and a user.

Many different kinds of specifications:
- Requirements specification: agreement between end user and developer
- Design specification: agreement between system architect and the implementer
- Module specification: agreement between programmer using the module and programmer implementing the module
Uses of specifications

Many:

- Statement of user requirements
- Statement of the interface between the machine and the controlled environment
- Statement of the requirements for the implementation
- A reference point during product maintenance

For today, we focus on statements of user requirements; however techniques we will study apply generally
Quality attributes of a specification

What makes a *good* specification?

Three major qualities:
- Clarity (i.e., understandability, lack of ambiguity)
- Consistency
- Completeness

We will now study:
- Characteristics of a *bad* specification
- Model-based techniques for *designing in* clarity and consistency
7 deadly sins of specifier

Noise: text that contains no relevant information;
   – redundancy
   – remorse

Silence: unspecified feature; lack of definition

Overspecification: solution/program details not problem description

Contradiction: incompatible specifications

Ambiguity: multiple possible interpretations

Forward reference: mention of a feature before it is defined

Wishful thinking: a requirement that cannot be validated
Case study: Text formatter
(Naur’s specification)

Given a text consisting of words separated by BLANKS or by NL (new line) characters, convert it to a line-by-line form in accordance with the following rules:

1. line breaks must be made only where the given text has BLANK or NL;
2. each line is filled as far as possible, as long as
3. no line will contain more than MAXPOS characters.
Exercise

Find three problems with Naur’s specification
Problems with Naur’s specification

No definition of "line"; no statement of the relationship of "line" to the new line character.

If the input text does not end with an NL, then (presumably) it is not a line; hence, it is meaningless to talk of it being converted on "a line-by-line" basis.

More generally, there is no description of the structure of the input file; no definition in terms of lines. There is also no definition of the output file structure.

There is no prescription about what to do about multiple blanks and at least two interpretations are possible. There is no rule about zero length words and in fact no definition of "word".
Problems (continued)

Does not prescribe what to do with multiple newlines.

Nothing is stated about restrictions on the input file; specifically, what to do if an input line has a word containing more than MAXPOS characters.

Does not prescribe what to do if the first characters in the input are one or more BLANKs.

There is no rule considering the case where multiple BLANKs end the file or come just before a NL that ends the file.
Case study: Text formatter  
(Gerhart/Goodenough specification)

The program's input is a stream of characters whose end is signaled with a special end-of-text character, ET. There is exactly one ET character in each input stream.

Characters are classified as:

– break characters--BL (blank) and NL (new line);
– nonbreak characters--all others except ET;
– the end-of-text indicator--ET.

A word is a nonempty sequence of nonbreak characters. A break is a sequence of one or more break characters. Thus, the input can be viewed as a sequence of words separated by breaks, with possibly leading and trailing breaks, and ending with ET.
Gerhart/Goodenough spec (cont)

The program's output should be the same sequence of words as in the input with the exception that an oversized word (i.e., a word containing more than MAXPOS characters, where MAXPOS is a positive integer) should cause an error exit from the program (i.e., a variable, Alarm, should have the value TRUE), up to the point of an error, the program's output should have the following properties:

- A new line should start only between words and at the beginning of the output text, if any.
- A break in the input is reduced to a single break character in the output.
- As many words as possible should be placed on each line (i.e., between successive NL characters).
- No line may contain more than MAXPOS characters (words and BLs).
Exercise

Find four problems with the Gerhart/Goodenough specification
Problems with Gerhart spec

Use of both of the phrases "non-empty" and "one or more". Why not avoid potential confusion by using just one phrase? Same holds for "stream" and "sequence".

"Output text, if any" is an example of remorse. A qualification to the definition of output text appears long after it is used.

"line" is used before it is defined. The definition is incorrect (it defines a line as being between NL's which ignores text before the first NL). The meaning of NL and its relation to the concept of line is left implicit.

"Alarm" is a program concept instead of a problem concept. Behavior of Alarm is not specified if MAXPOS is never exceeded. The "point of error" is not defined.

Problems w/ Gerhart spec (cont)

Contradictory definitions of input ("stream of characters" and 
"sequence of words separated by breaks").

ET is a machine dependent (program domain) concept. It is 
used three times before it is defined. The output file does 
not contain ET which is either a bug in the spec or a 
significant non-uniformity.

The phrase "trailing blanks ending with ET" is confusing.

MAXPOS is used before it is defined.
Software specification

(2) Unambiguous specifications

Topics:

– Techniques for addressing ambiguity problems
– Use of formal methods to “design” a good spec
– Companion paper:
Ambiguity

Tends to arise from use of undefined concepts
  – Unfair to say specifier “doesn’t know what he/she is talking about”
  – Usually, concepts are clear but indistinct

Ambiguity is resolved through analysis and careful definition, thereby making clear concepts distinct

Claim: Mathematics provides a wealth of tools to support analysis and definition
How to design unambiguous specs

A description or definition (of anything) is a translation between the concept being defined and some other, hopefully simpler or more familiar, concepts.

Simpler concepts must be unambiguous:
- So mathematical constructs are often used
- E.g., sets, sequences, functions, and relations.

Approach:
1. Define low-level (fundamental) concepts first using enough precision to avoid ambiguity.
2. Build up the more complex concepts in terms of the lower level ones.
Functional specifications

Describe the relationship between the inputs to a software system and the outputs.

Can be expressed as a table with two columns.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>
Observations (functional specs)

The output column may always contain duplicates. If the input column contains duplicates then the program is called non-deterministic.

If either the input or the output is composite, then there may be more than two columns in the table. In any case, the table can be divided into two parts: those columns describing the input and those describing the output.
Functional spec of text processing

Primitive concepts:
- CHAR, undefined other than it is a set that contains at least the two elements \textit{blank} and \textit{new\_line}.
- BREAK\_CHAR is defined to be the set \{\textit{blank}, \textit{new\_line}\}.

Both the input and the output are sequences of CHAR.

Key ideas:
- Construct sets of subsequences that are related to the input by various rules;
- Select only those subsequences that obey other rules;
- Any element of this set (of subsequences) is a valid output of the program.
# Examples (MAXPOS=10)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>“a brown fox jumped over the fence at dawn”</td>
<td>“a brown fox jumped over the fence at dawn”</td>
</tr>
<tr>
<td>“O say can you see by the dawn’s early light”</td>
<td>“O say can you see by the dawn’s early light?”</td>
</tr>
</tbody>
</table>
Question

What would the input:

“What is the modern name of the ancient Roman capital Constantinople?”

map to?
Meyer’s observation

Text processing problem is difficult because it contains three non-orthogonal aspects:

1. Reducing breaks to a single break character;
2. Making sure that no line has more than \text{MAXPOS} characters; and
3. Filling these lines as much as possible

Solution: Formally specify each aspect in isolation; then combine these (partial) specifications to get a specification of the whole problem.
From formal to natural language

Formal spec provides a lot of hints on how to write the natural language spec.

Example:

– Should refrain from trying to define concept of a “line in the text”, which is tricky to specify precisely
– Rather, should focus on “maximum line length”, which is defined even if text contains only new_lines
Meyer’s final specification

Given are a nonnegative integer MAXPOS and a character set including two break characters, blank and new_line.

The program shall accept as input a finite sequence of characters and produce as output a sequence of characters satisfying the following conditions:

– It only differs from the input by having a single break character wherever the input has one or more break characters
– Any MAXPOS + 1 consecutive characters include a new_line
– The number of new_line characters is minimal

If (and only if) an input sequence contains a group of MAXPOS+1 non-break characters, there exists no such output. In this case the program shall produce the output associated with the initial part of the sequence up to and including the MAXPOS-th character of the first such group and report the error.