CSIT113 Problem Solving

UNIT 1
INTRODUCTION

UNIVERSITY OF WOLLONGONG AUSTRALIA

SIM GLOBAL EDUCATION

Overview

- Problem classification
- Basic framework for solving problems
- Algorithms
- Problem solving strategies

Q

Introduction: What is a problem?

Is this a problem?
- What is 3+4×5?

Is this a problem?
- Given the distribution of rain in the last week, what is the probability of flooding in building 3?

Is this a problem?
- You are sitting in a lecture and you want to be on the beach.

Q

Introduction: Some definitions.

- "A problem exists when the goal that is sought is not directly attainable by the performance of a simple act available in the animal's repertory…"

  Thorndike 1989
- "A person is confronted with a problem when he wants something and does not know immediately what series of actions he can perform to get it"

  Newell & Simon 1972
- "Whenever there is a gap between where you are now and where you want to be, and you don't know how to cross that gap, you have a problem"

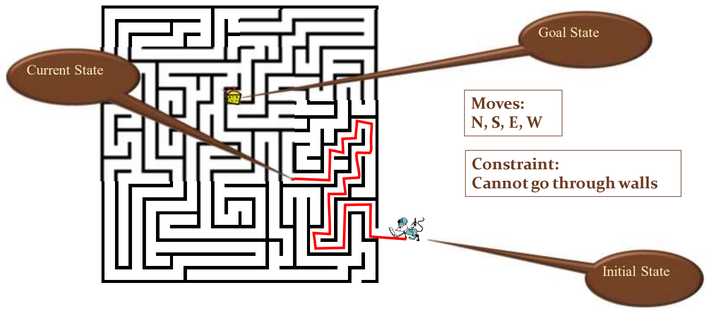
  Hayes, 1980

Introduction: Problem components.

- All problems have:

A start state	Where you begin.
A current state	Where you are now.
A goal state	Where you want to be.
Operators	To move between states.
Constraints	Limits on moves or solution.

A problem exists when there is a 'gap' between the current state and the goal state!



Your turn.

- Identify the start and goal states, and the operators and their constraints in the following problem:
  - What is 3+4×5?

Answer:

start state:	the problem as given
goal state:	the numerical answer
operators:	addition and multiplication
constraints:	the rules of arithmetic

  - What is the 'gap' in this problem?

Your turn.

- Identify the start and goal states, and the operators and their constraints in the following problem:
  - You are sitting in a lecture and you want to be on the beach.

Answer:

start state:
goal state:
operators:
constraints:

  - In this problem, the answer is less clear.

Types of problem.

- Problems can be classified in a number of different ways.
- Know what type of problem you have.
- This helps you better understand the problem, and what you might need to do to solve it.
  - Or whether it is even worth trying.

Well vs. Ill defined problems.

1. If two fair six-sided dice are rolled, what is the probability of obtaining a total of 12?
2. Plan a meal for a guest.

- Well defined problem:
  - All the information necessary to solve the problem is either explicitly given or can be easily inferred
- Ill defined problem:
  - There is uncertainty in either the initial state, the permissible operations/actions, or the final state.

Knowledge rich vs. poor problems.

1. Devise an efficient memory management scheme for a multi-processor computer with job sizes distributed according to a normal distribution with μ=500MB and σ=128MB.
2. Starting with the sequence of pieces given below:

● ● ● ● _ ○ ○ ○ ○

rearrange them so that they end up in the following sequence:

○ ○ ○ ○ _ ● ● ● ●

A dot can move to an empty space adjacent to it or jump over only ONE dot of the other colour into an empty space. White dots can only move to the left and brown dots to the right.

Knowledge rich vs. poor problems.

- Knowledge Rich Problems:
  - Require prior, specialized or domain-specific knowledge to solve.
- Knowledge Poor Problems:
  - Do not require specialized knowledge. Can be solved using strategies and methods that apply to many types of problems.
- These are also known as **domain-specific** and **domain-general** problems.

Q

Hard vs. Easy problems.

1. What are the factors of 9020779199?
2. What is 94331×95629?

- How hard a problem depends on how much work you need to do to reach the goal.
- Problems have varying levels of **complexity** which is a measure of how hard they are to solve.
- We will look more closely at this later in the subject.

Q

Important Problem Types

Classified from their characteristics, the most important problem types in CS/IT are:

- Sorting
- Searching
- Graph problems
- Combinatorial problems
- Numerical problems
- String processing
- Geometric problems

Q

Introduction: What is problem solving?

- "the bridging of the gap between an initial state of affairs and a desired state where no predetermined operator or strategy is known to the individual"
  - Ollinger, M. & Goel, V. "Problem-Solving." in V. Goel, & A. von Müller (Eds), Towards a Theory of Thinking. Springer. Goel, V. 2010
- ... involves pursuing a goal state
- ... finding a way to reach it
- ... requires mental activity.

Polya's* Problem Solving Method.

Polya's method has four steps:

1. Understand the problem: Make sure you understand what the problem is asking.
2. Plan a strategy for solving the problem.
3. Execute your strategy, and revise it if necessary.
4. Check and interpret your result.

In this subject we will use/discuss this method, although we will often not explicitly refer to it!

* George Polya "How to Solve It", Princeton University Press, 1945

A Simple Problem.

- A butcher is six foot, four inches tall and wears size 14 shoes.
- What does he weigh?
- The butcher weighs meat!
- Make sure the question is asking what you think it is!
- Ok, this is a tricky question but it makes an important point.

Understanding the problem.

- Read the problem very carefully.
- Take note of any data, or information, that is given. E.g. numbers, quantities, names, etc.
- From the data given:
  - Identify the start state
  - Identify the goal state
    - What is it you have to determine? (unknowns)
  - Identify any constraints
    - What are you not allowed to do?
  - Identify any operators
    - What can you do?
- If the problem is not clear restate it in different ways to clarify it.
- This is just the first step to solving a problem!

Some practice problems: approach with caution.

1. Anna had six apples and ate all but four of them. How many apples were left?
2. If there are 12 one-cent stamps in a dozen, how many two-cent stamps are there in a dozen?
3. If Mr. Howard's rooster laid an egg in Mr. Bush's yard, who owns the egg?
4. A lady did not have her driver's license with her when she failed to stop at a stop sign and then went three blocks down a one-way street the wrong way. A policeman saw her, but he did not stop her. Explain.

Algorithms.

- A lot of the problems we encounter in computing can be specified using algorithms.
- An algorithm is a systematic description of how to solve a problem.
- Algorithms have many aliases:

Problem	Algorithm
Knit a sweater	Pattern
Bake a cake	Recipe
Drive to Sydney	Directions

What is an algorithm?

- A method for accomplishing a task (such as performing a calculation or solving a problem).
- A finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time.
- A precise description of the steps required to go from an initial state to a final state.

Properties of an algorithm.

- An algorithm must be:
  - Finite – it should terminate.
  - Complete – it should specify every step that must be performed
  - Correct or Effective – it should give you the right answer or get to the desired goal.
  - Precise – it should be clear and specific enough that it can be followed.
- An algorithm should be:
  - Efficient – it should arrive at the answer in a reasonable time or with a reasonable amount of effort.
  - General – it should solve more than just the specific problem being considered.

Q

Correctness vs. Efficiency.

- An algorithm is correct if it comes up with the solution.
- Sometimes we have no efficient way to do this.
- In this case we may be satisfied with an algorithm that gives us a ‘good enough’ or ‘near-enough’ solution.
- Lots of real world problems fall into this category.

Q

Efficiency and Complexity.

- Can we estimate how efficient an algorithm is?
- The efficiency of an algorithm is expressed in terms of complexity
  - Time complexity
  - Space complexity
  - Communication complexity
- The lower bound complexity for an algorithm to solve a particular problem is the complexity of that problem.
  - In general, no algorithm to solve a problem can be less complex than an algorithm to check a solution.

Q

Generality.

- A good algorithm should be general
  - An algorithm that works in all cases of a problem is better than an algorithm that only works in certain cases.
  - An algorithm that solves a set of related problems is better than an algorithm that only solves a specific problem.
  - We will talk a bit more about generality when we discuss abstraction and preconditions.

Q

Classifying algorithms – One Common Way:

- Algorithms can be categorised by the domain to which they apply:
  - graph algorithms
  - string algorithms
  - sorting algorithms
  - cryptographic algorithms
  - compression algorithms
  - etc.

Q

Classifying algorithms – Another Common Way:

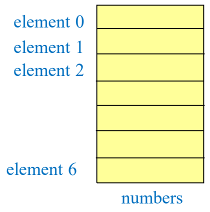
- Algorithms can be categorised by the approach which they apply:
  - greedy
  - divide and conquer
  - recursive
  - branch and bound
  - genetic
  - heuristic
  - etc.

Creating algorithms.

- So, how do you create an algorithm?
  - Solve the problem.
  - Write down the steps taken in solving the problem.
  - Pseudocode is used to specify (write) algorithms: next, we shall introduce sequence that will be used in writing algorithms before introducing pseudocode briefly.
- The first step may be a bit tricky!
- This is the main topic of this subject.
- As we will see in future weeks there are a number of broad principles and approaches which are useful in solving all sorts of problems.
- We often refer to these as strategies.

What is an sequence

- An sequence is a fixed-size, sequenced collection of elements of the same data type



Q: how many elements in this sequence?

There are 7, indexed from 0 to 6

- The **length or size** of an sequence is the number of elements in the sequence.
- Each element in the sequence is identified by an integer called **index**, which indicates the position of the element in the sequence.
- An **sequence's index** starts from **0** and goes up to the **sequence's length minus 1**

An sequence of Nine Integers

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
ages	10	34	9	1	2	10	3	89	3

ages[0] = ?	ages[0] = 10
ages[5] = ?	ages[5] = 10
ages[7] = ?	ages[7] = 89
ages[2] = ?	ages[2] = 9
ages[9] =?	ages[9] =illegal
ages[11]=?	ages[11]=illegal

Pseudocode

- High-level description of an algo.
- More structured than English prose
- Less detailed than a program
- 3 basic control constructs: sequence, selection and iteration
- Control constructs must be reflected clearly by some standard conventions (many options are available)

Example: find the largest element in an sequence

Input sequence A of n integers
Output largest element in A

```
sequenceMax(A, n) {
   Max = A[0]
   for (i = 1 to n – 1)
      if A[i] > Max then
         Max = A[i]
   return Max
}
```

	0	1	2	3	4
A	9	10	25	34	6

Max: 34

Pseudocode Details

- Control flow:
  - **if** … … [**else** …]
  - **while** ……..
  - **for** … …
- Algorithm declaration:
  - Input …
  - Output …
  - **Algorithm_*Name*(*arg*, *arg*,…)**
- Calling another algo:
  - ***Algo_Called* (*arg*, *arg*,…)**
- Return value:
  - **return *expression***

- Some common notation:

=	Assignment	**x = b**
==	Equality testing	**if x = = b**
n^2	Superscripts and other mathematical formatting allowed	
$\lfloor x \rfloor$	The largest integer f such that $f \le x$	
$\lceil x \rceil$	The smallest integer c such that $c \ge x$	

- Other Mathematical formatting are also allowed

Pseudocode: The return statement

```
return expression
```

- The expression under the return statement can only be a single variable or single expression with the following meaning:
  - ✓ Single variable: The value of the variable upon the execution of the statement is returned.
  - ✓ Single expression: The value of the expression upon the execution of the statement is returned.
- Regardless of where the return statement is placed, immediately after the execution of the statement, **the execution STOP and EXIT.** No further execution of any statement in the same procedure containing the return statement.

Pseudocode: Selection

```
if (condition)
   action1
```

```
if (condition)
   action1
else
   action2
```

If an action has more than one statement, enclose all the statements by a pair of braces, {}.

```
Example:
if (b > x) // if b is larger than x, update x
    x = b
if (c > x) // if c is larger than x, update x
    x = c
return x

Example:
if ( x ≥ 0) {
      x = x + 1
      a = b + c
}
```

*Pseudocode: while loop

```
while (condition)
    action
```

	0	1	2	3	4
s	-3	20	450	89	90

✓ **If *condition* is true, action is executed.**
✓ **The process is repeated until condition becomes false**

```
//Find the max value in an sequence s of n numbers using a while loop
sequence_max1(s, n) {
    large = s[0]
    i = 1
    while (i < n) {
      if (s[i] > large) // larger value found
         large = s[i]
      i = i + 1
    }
    return large
}
```

Pseudocode: for loop

```
for (var = init to limit)
    action
```

0	1	2	3	4
-3	20	450	89	90

s

✓ **The variable *var* is first set to the value *init*. If *var ≤limit*, action is executed and 1 is added to *var*.**
✓ **The process is repeated until *var > limit***

```
// Find the max value in an sequence s of n numbers using a for
loop
sequence_max2 (s, n) {
    large = s[0]
    for (i = 1 to (n-1) ) {
       if (s[i] > large) // larger value found
         large = s[i]
    }
    return large
}
```

Pseudocode: for loop

```
for (var = init downto limit)
   action
```

- ✓ Action is executed as long as *var* ≥ *limit*, and 1 is subtracted from *var* *after each loop iteration*
- ✓ The process is repeated until *var* < *limit*

Write an algorithm that returns the *index* of the last occurrence of the value *key* in the sequence s[0], . . . , s[n-1]. If *key* is not in the sequence, the algorithm returns the value 99999.

```
find_last_key (s, n, key) {
    for (i = (n-1) downto 0 ) {
      if (s[i] == key ) // key value found
         return i
    }
    return 99999
}
```

Further Example on Algorithm Execution 1

What is the value returned by findTarget([3, 15, 7, 2], 4)?

```
findTarget(s, n) {
        EleFound = s[0]
        for (i = (n-1) downto 0 ) {
                if (s[i] > EleFound)
                        EleFound = s[i]
                return EleFound
        }
}
```

Explanation:
In the execution of findTarget([3, 15, 7, 2], 4):

Initially, EleFound is set to s[0] =3.
Then, it executes the for loop with i starts from (4-1) = 3 down to 0 as follows:
1st iteration: i = 3.
First, as s[3] = 2 > 3 is false, EleFound remains at 3.
Second, the return statement is executed to return 3 and stop the whole execution.

Further Example on Algorithm Execution 2

What is the value returned by Find9([15, 3, 7, 9], 4)?

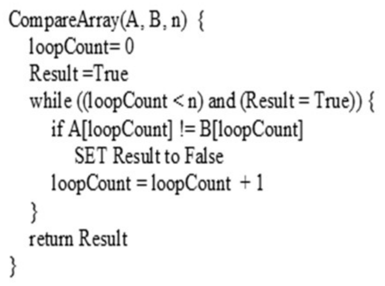
```
Find9(s, n) {
    EleFound = s[0]
    for (i = (n-1) downto 0 ) {
        if (s[i] < EleFound)
        EleFound = s[i]
    }
    return EleFound
}
```

Explanation:
In the execution of Find9([15, 3, 7, 9], 4):

Initially, EleFound = s[0] = 15.
Then, it executes the for loop with i starts from (4-1) = 3 down to 0:
1st iteration: i = 3. As s[3] = 9 < 15 is true, EleFound is set to s[3] = 9.
2nd iteration: i = 2. As s[2] = 7 < 9 is true, EleFound is set to s[2] = 7.
3rd iteration: i = 1. As s[1] = 3 < 7 is true, EleFound is set to s[1] = 3.
4th iteration: i = 0. As s[0] = 15 < 7 is false, EleFound remains unchanged at 3. Now, for loop execution finished.
Next, it executes the return statement after the for loop, and return Elefound = 3 and stop.

Further Example on Algorithm Execution 3

The following algorithm correctly checks the equality of two sequences A and B:



```
CompareArray(A, B, n) {
  loopCount= 0
  Result =True
  while ((loopCount < n) and (Result = True)) {
    if A[loopCount] != B[loopCount]
      SET Result to False
    loopCount = loopCount + 1
  }
  return Result
}
```

Figure 11. A simple algorithm

Hint for Explanation:
(1) Test this algo by checking unequal array, e.g., CompareArray([3, 4, 7], [999, 888, 7], 3), …..
It all returns False.
(2) Test this algo by checking equal array, e.g., CompareArray([3, 4, 7, 9, 0], [3, 4, 7, 9. 0], 5), …..
It all returns True.

Further Example on Algorithm Execution 4

The following algorithm wrongly checks the equality of two sequences A and B: just based on the equality of the last element

```
CompareArray(A, B, n) {
   loopCount= 0
   Result =True
   while ((loopCount < n) {
      if A[loopCount] != B[loopCount]
         Result = False
      else
         Result = True
      LoopCount = LoopCount + 1
   }
   return Result
}
```

Hint for Explanation:
Test this algo by finding CompareArray([3, 4, 7], [999, 888, 7], 3).
It returns True.
Hence, wrong.

Q

Problem Solving Strategies.

- Explore the problem space
  - Consider potential states of the problem until you find one that is a solution
  - This covers a range of strategies including
    - brute-force
    - trial and error
    - backtracking
    - depth-first search
    - etc.

Q

Problem Solving Strategies.

- Analogy
  Is this like a problem you have solved before?
  - A tourist wants to convert 100 US dollars to Australian dollars. If the rate is 1.3 AUD to the USD, how many Australian dollars will he get?
  - A car stops after 100 seconds. If it travelled at an average speed of 1.3 m/s. How far has it travelled?
  - A car moving at an average speed of 100km/h travels for 1.3 hours. How far has it gone?

Q

Problem Solving Strategies.

- Reduction
  Can you reduce it to a problem you have solved before?
  - A team of three runners competes in a race, each person runs for 3 minutes. If the first person runs at 3m/s, the second person runs at 4m/s and the third person runs at 5m/s, how far have they run?
- This is not always a good idea.

Problem Solving Strategies.

- Research
  Has someone else already solved a problem like this?
  - There is no need to reinvent the wheel
  - Read widely
  - Discuss with others
  - Consider the type of problem you are dealing with for some hints on where to look
  - Note:
    - Typically not an appropriate technique in assignments
    - Always give credit

Problem Solving Strategies.

- Abstraction
  Are there any high-level patterns or principles?
  - If you can identify and represent the key aspects of a problem, it is much easier to solve
  - This is an important strategy that covers lots of aspects
    - Symmetry
    - Pattern Recognition
    - Maths
    - Notation
    - etc.

Problem Solving Strategies.

- Solve a simpler problem
  Can solving a simplified problem help solve the original problem?
  - The solution to the simple problem may give you
    - insight into the original problem
    - or a partial solution
  - Covers a range of different techniques, depending on how you simplify

Problem Solving Strategies.

- Solve a simpler problem - reduce the number of constraints
  - Informal technique:
    - throw away some constraints, solve the problem, work out how to reintroduce the constraints
  - Formal technique:
    - Branch and bound

Q

Problem Solving Strategies.

- Solve a simpler problem - Solve part of the problem
  - Start close to the goal, work backwards
  - Start from both ends, work towards the middle
  - Break the problem into sub-problems and solve each sub-problem
  - Formal techniques
    - divide and conquer
    - functional decomposition / modularisation

Q

Problem Solving Strategies.

- Solve a simpler problem - Make the problem smaller
  - Instead of solving a problem involving 1000 people, solve it for 1 person, then 2 people, then 3…
  - Identify a pattern
  - Use that to solve your original problem
  - Formal techniques:
    - Induction
    - Recursion

Q

How do I learn to solve problems?

- Do CSIT113 (hopefully this will help).
- Practice.
- Practice some more.
- Practicing will enable you to:
  - Identify similarities between problems.
  - Master techniques and variations of them.
  - Gain confidence in your ability to solve problems.
    - It is less likely you will be put of by something that looks tricky!

Q

After you solve a problem.

- Don't just solve the problem, think about how you solved the problem
- This is a critical factor in coming up with an algorithm
- Think about how you solved it and what type of problem it is
- This helps improve your strategy and allows you to more easily recognise strategies and when to use them
- Even if you can't solve a problem, think about what is stopping you from solving it
  - lack of knowledge? sub-problem? not solvable?

Q

After you solve a problem.

- Often when solving a problem, you are too busy trying to get the answer to focus on how you are thinking or what you are learning
- You should reflect on what strategies you used, how well they worked, what types of problem you encountered, whether you need more practice, etc.

Q

After you solve a problem.

- Think about how you solved the problem and then explain it to someone else
  - Your friend, grandmother, dog, computer
  - Make use of tutorials for this
- Trying to explain your thinking helps you understand it better
- Writing an algorithm requires you to clearly express your strategy

Q

Some sample problems:

Student: All three of my sons celebrate their birthday today. Can you tell me how old each of them is?

Lecturer: Sure, but you will have to tell me something about them.

Student: The product of their ages is 36.

Lecturer: … I'll need more information.

Student: The sum of their ages is equal to the number of windows in that building (13).

Lecturer: Good, but I still need an additional hint to solve the puzzle.

Student: My oldest son has blue eyes.

Lecturer: Ah, the ages of your sons are …

Q

What do we know?

- What does each piece of the story tell us?

Student: All three of my sons celebrate their birthday today. Can you tell me how old each of them is?

There are three sons

What do we know?

- What does each piece of the story tell us?

  Student: The product of their ages is 36.

Age of Son 1	Age of Son 2	Age of Son 3	
36	1	1	
18	2	1	
12	3	1	
9	4	1	
9	2	2	
6	6	1	
6	3	2	
4	3	3	

What do we know?

- What does each piece of the story tell us?

  Student: The sum of their ages is equal to the number of windows in that building (13).

Age of Son 1	Age of Son 2	Age of Son 3	Sum of Ages
36	1	1	38
18	2	1	21
12	3	1	16
9	4	1	14
9	2	2	13
6	6	1	13
6	3	2	11
4	3	3	10

What do we know?

- What does each piece of the story tell us?

  Lecturer: Good, but I still need an additional hint to solve the puzzle.
- Why can't he solve it yet?

Age of Son 1	Age of Son 2	Age of Son 3	Sum of Ages
36	1	1	38
18	2	1	21
12	3	1	16
9	4	1	14
9	2	2	13
6	6	1	13
6	3	2	11
4	3	3	10

What do we know?

- What does each piece of the story tell us?

  Student: My oldest son has blue eyes.
- There is an oldest son!

Age of Son 1	Age of Son 2	Age of Son 3	Sum of Ages
~~36~~	~~1~~	~~1~~	~~38~~
~~18~~	~~2~~	~~1~~	~~21~~
~~12~~	~~3~~	~~1~~	~~16~~
~~9~~	~~4~~	~~1~~	~~14~~
9	2	2	13
6	6	1	13
~~6~~	~~3~~	~~2~~	~~11~~
~~4~~	~~3~~	~~3~~	~~10~~

Q

What do we know?

- We now have the only possible answer!
- This solution used a combination of strategies
  - Brute-force, to get the initial table
  - Logic, to remove options

Age of Son 1	Age of Son 2	Age of Son 3	Sum of Ages
~~36~~	~~1~~	~~1~~	~~38~~
~~18~~	~~2~~	~~1~~	~~21~~
~~12~~	~~3~~	~~1~~	~~16~~
~~9~~	~~4~~	~~1~~	~~14~~
9	2	2	13
~~6~~	~~6~~	~~1~~	~~13~~
~~6~~	~~3~~	~~2~~	~~11~~
~~4~~	~~3~~	~~3~~	~~10~~

Q

Another example:

- There are three boxes:
  - One contains only apples.
  - One contains only pears.
  - One contains both apples and pears.
- Each box is labelled:
  - No box has the correct label.
- You are allowed to examine **one** piece of fruit from **one** box.
- Can you label all the boxes correctly?

Q

What do we know?

- There are only two possible arrangements that fit the description:

Label	"Apples"	"Pears"	"Apples and Pears"
Arrangement 1	Pears	Apples and Pears	Apples
Arrangement 2	Apples and Pears	Apples	Pears

- How can we tell them apart with a single piece of fruit? Try the box labelled with "Apples and Pears"

Q

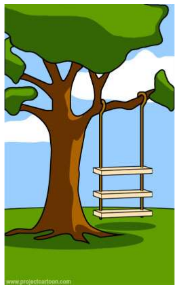
It's not all algorithms.

- Though algorithms are very important for solving problems in Computer Science and Information Technology areas, not every problem can be solved by constructing an algorithm.
- Often, problems arise because of poor communication.
- Consider the example of the garden swing…

Q

The garden swing.

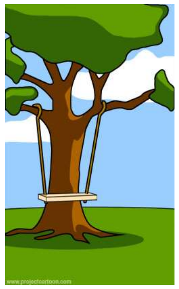
What the customer described...



Q

The garden swing.

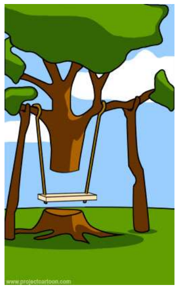
What the project leader heard...



Q

The garden swing.

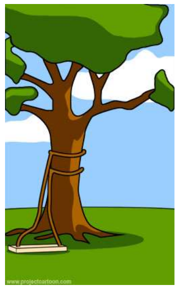
What the analyst designed...



Q

The garden swing.

What the programmer wrote...



Q

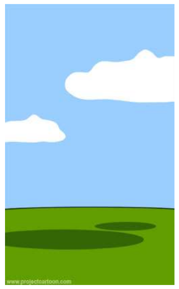
The garden swing.

What the beta testers got...

Q

The garden swing.

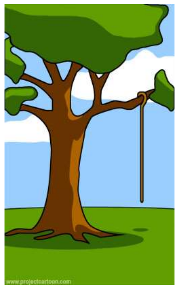
The documentation...



Q

The garden swing.

What was first installed...



Q

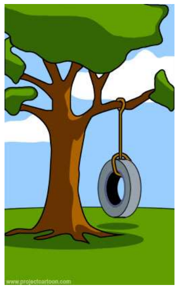
The garden swing.

What was finally installed...

Q

The garden swing.

What was needed!



Q

A final problem (for today).

- Consider a bench with enough room on it for three people.
- Two people are sitting on it.
  - One is in the middle and one is to the left.
- The question:
  - Who sat down first?
- Do we have enough information?
- What is involved?
  - Psychology.
  - Probability.
  - …?

Q

A final thought (for today).

- Clarke's Laws: (paraphrased)

1. When a distinguished but elderly lecturer states that something is possible, he is almost certainly right. When he states that something is impossible, he is very frequently wrong.
2. The only way of discovering the limits of the possible is to venture a little way past them into the impossible.
3. Any sufficiently advanced technology is indistinguishable from magic.

- If we don't understand the solution – it's magic!
- If we do understand the solution – it's technology!