# CSSE 490 Model-Based Software Engineering: Automatic Programming Perspectives

Shawn Bohner

Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# Learning Outcomes: MBE Discipline

***Relate Model-Based Engineering as an engineering discipline.***

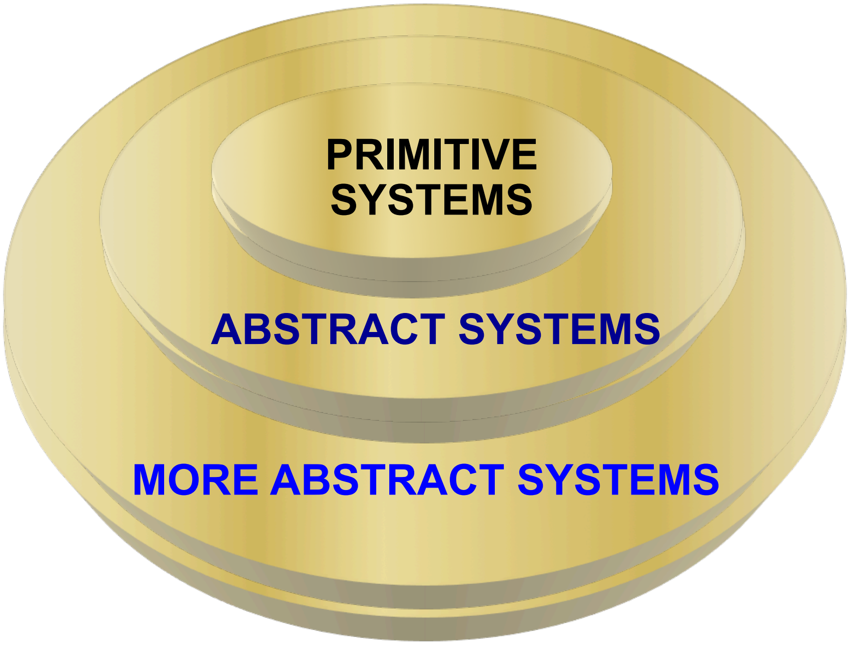
- Discussion of Milestone 3
- Introduce Automatic Programming
- Look at Assistant approach (if time)

# What would you say to the statement "Today's specification language becomes tomorrow's programming language?"

- Think for 15 seconds…
- Let's talk…

# Philosophy: Reliable Systems are Defined in Terms of Reliable Systems

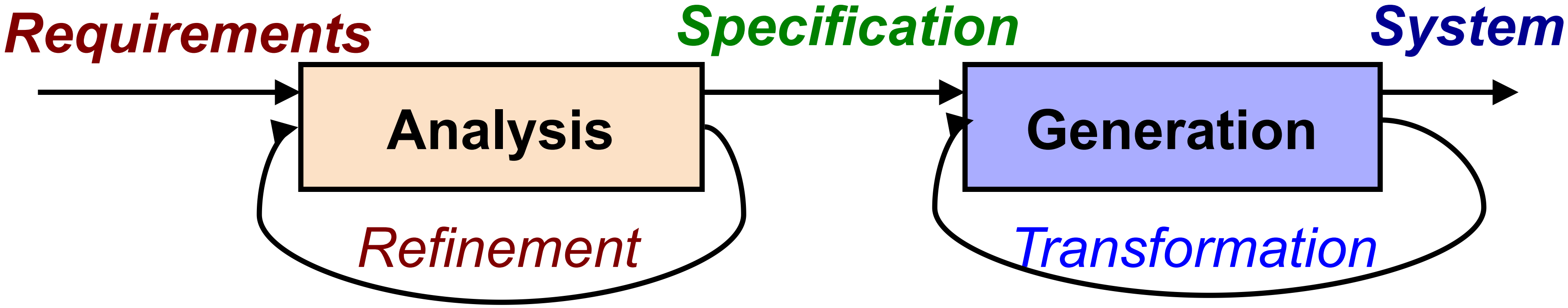
- Use only reliable systems
- Integrate these systems with reliable systems
- The result is a system(s) which is reliable
- Use resulting reliable system(s) along with more primitive ones to build new and larger reliable systems



*A recursively reliable and reusable process*

# Automatic Programming

- Getting software to write software
- Great idea, but turns out to be hard
- Should be easier than other tasks
  - □ But programming requires some strategy (i.e., cunning and guile ☺ )
  - □ Many human tasks difficult to automate



Requirements

Analysis

Refinement

Specification

Generation

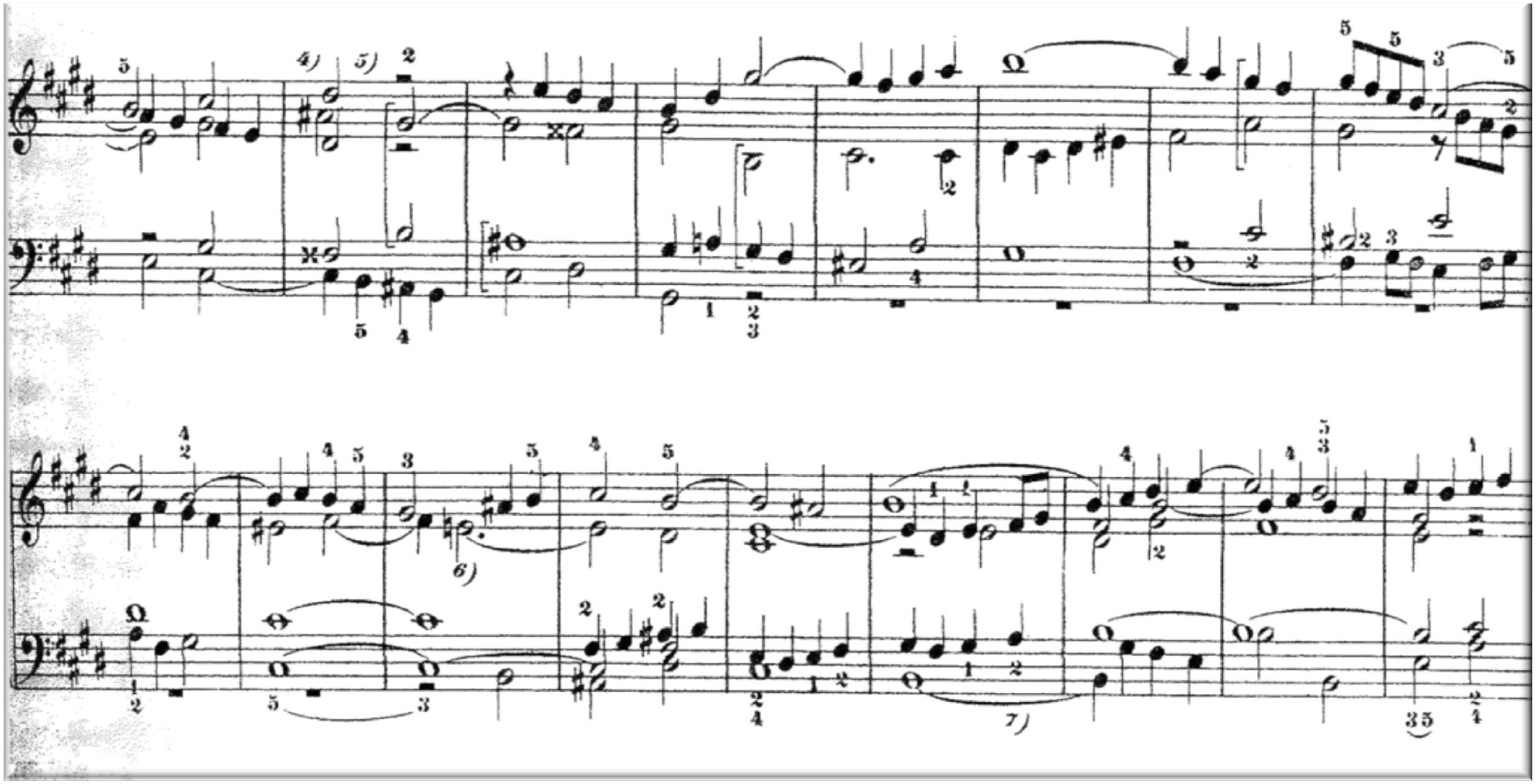
Transformation

System

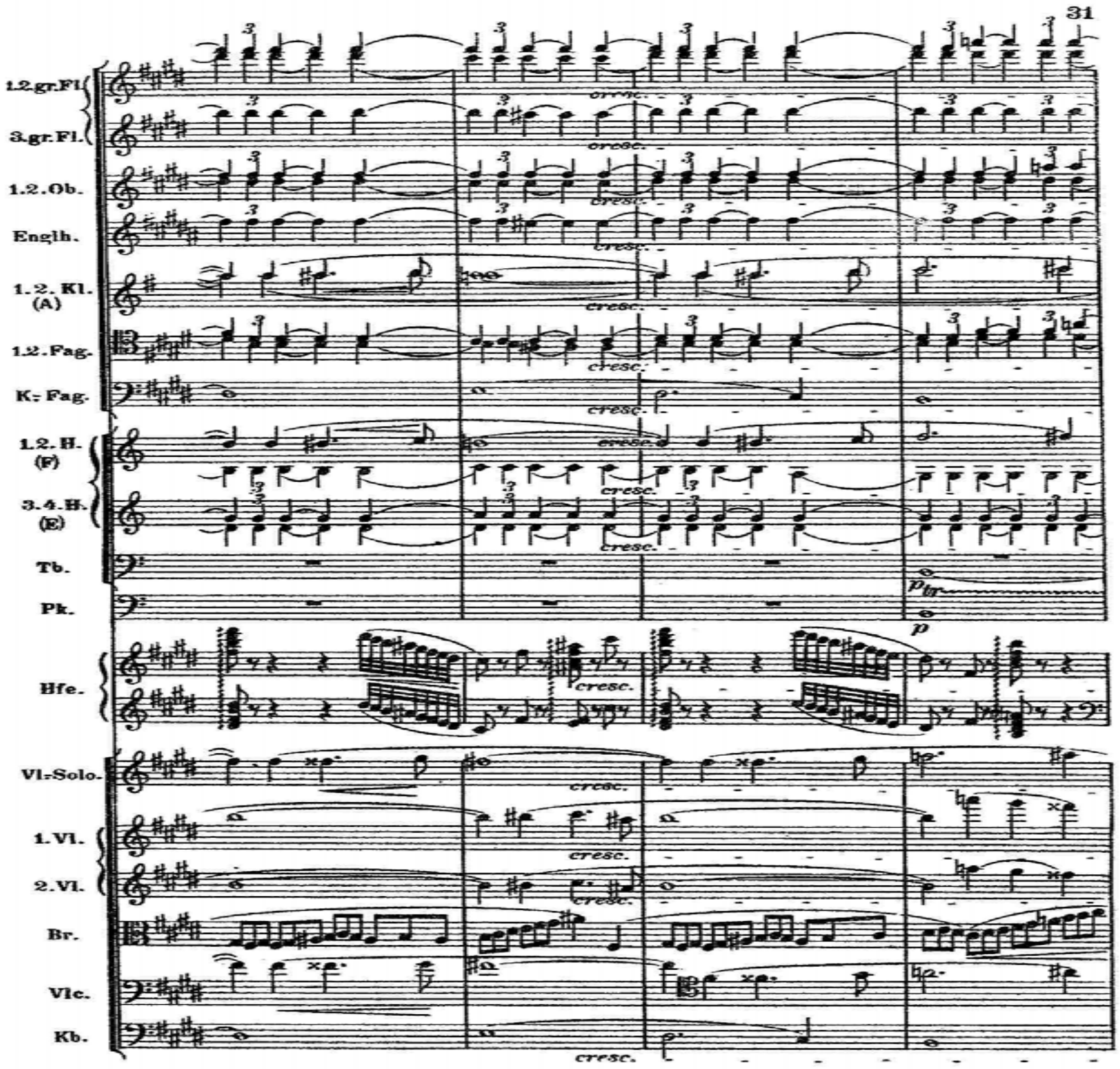
# Automatic Programming

- Oversold early on and under-delivered on promises
- So people began to avoid this area
- "Automated Programming" became words of warning
- Since then, the limitations have eased
  - □ Memory space
  - □ Knowledge representation
  - □ Transformation systems

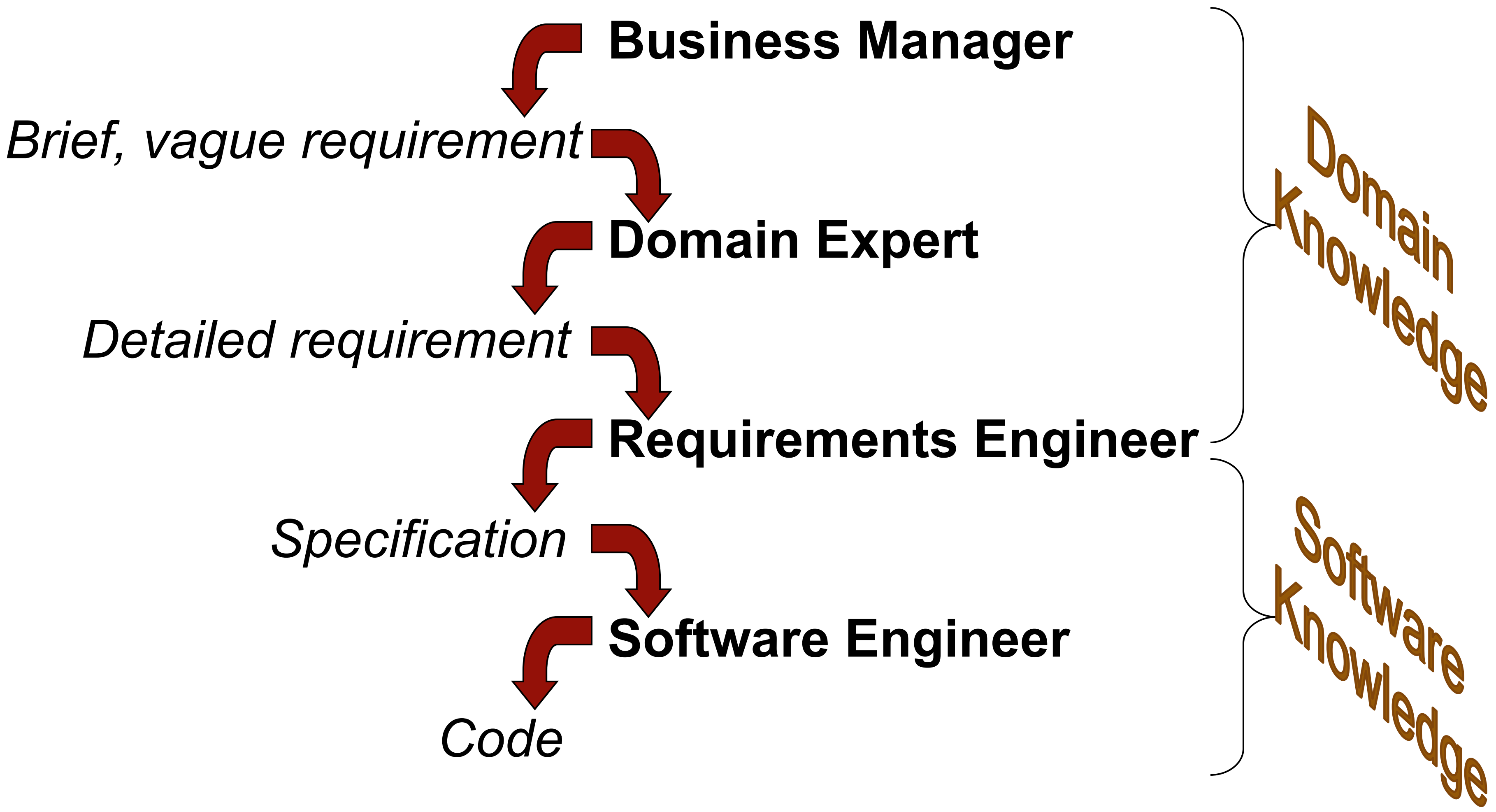
# Complexity: Intricacy (Bach)



# Complexity: Volume of Detail (Strauss)



# Automatable Programming Activities



**Business Manager**

*Brief, vague requirement*

**Domain Expert**

*Detailed requirement*

**Requirements Engineer**

*Specification*

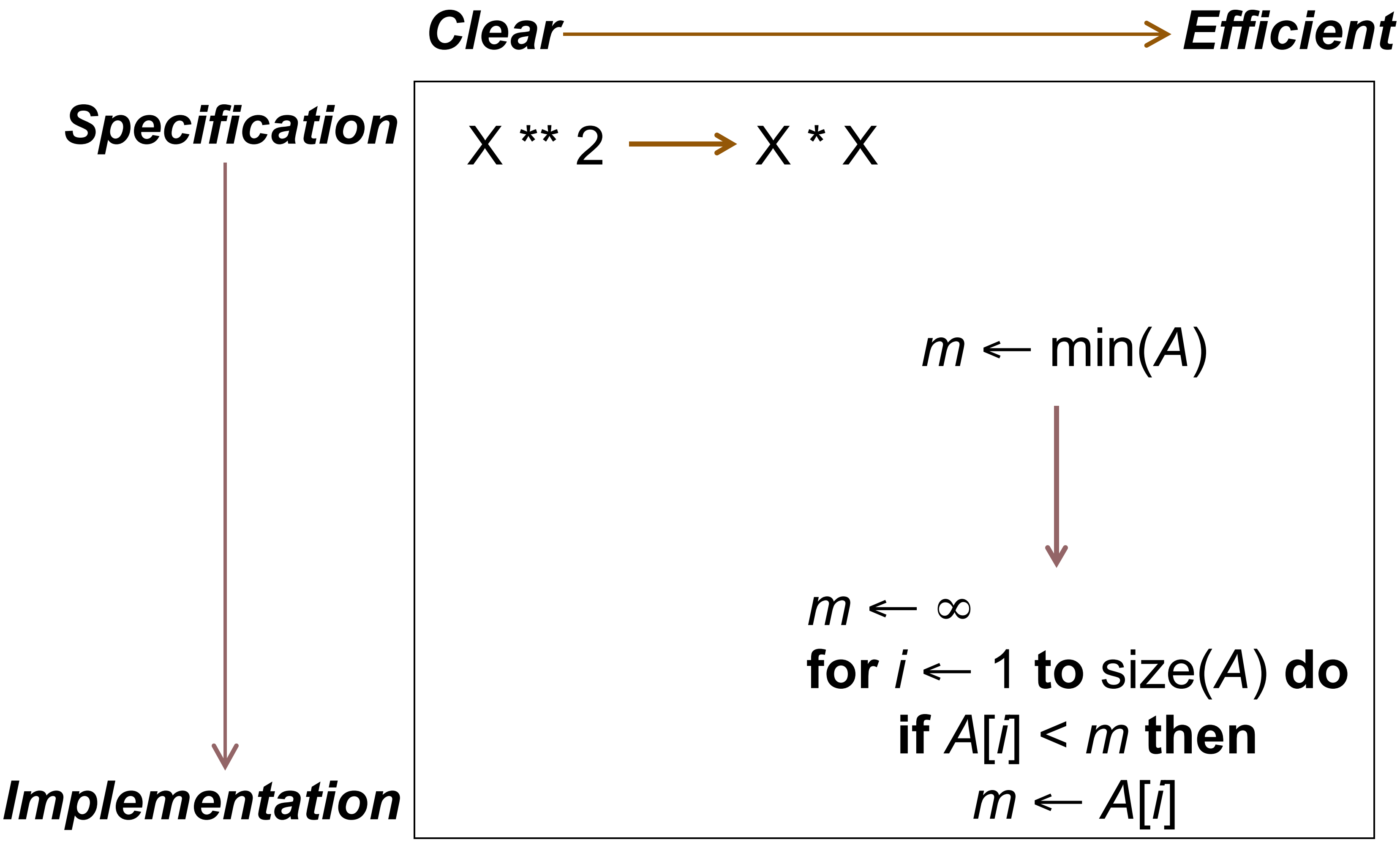
**Software Engineer**

*Code*

Domain Knowledge

Software Knowledge

# Transformational Approaches



*Clear* → *Efficient*

*Specification*

↓

*Implementation*

X ** 2 → X * X

$m \leftarrow \min(A)$

↓

```
$m \leftarrow \infty$
for $i \leftarrow 1$ to $\mathrm{size}(A)$ do
    if $A[i] < m$ then
        $m \leftarrow A[i]$
```

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# Natural Language Specification

*"The SystemX transmission times are entered into the schedule"*

↓

*"Each SystemX clock transmission times and transmission length is made a component of a new transmission entry which is entered into the transmission schedule"*

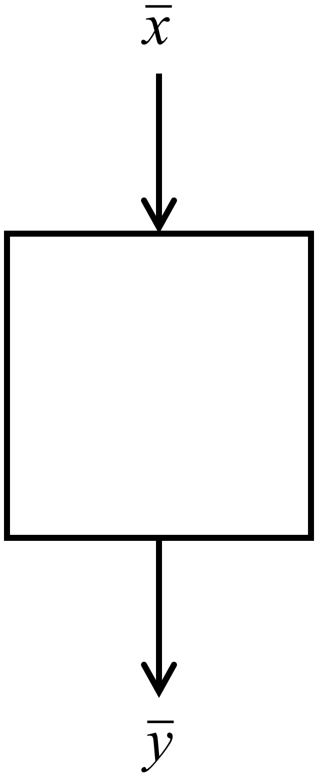
***Problem is informality***

# Deductive Synthesis

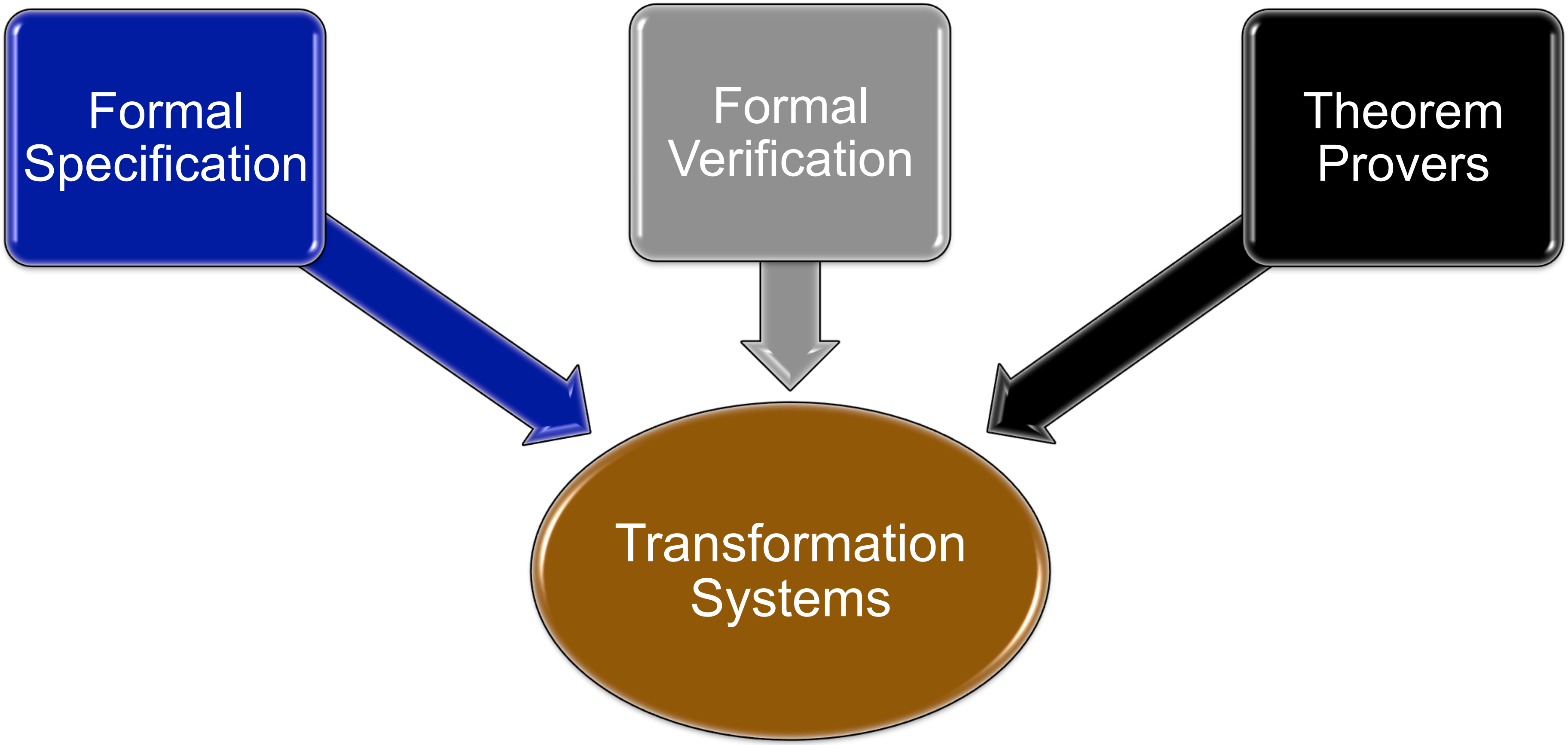
Precondition: $P(\bar{x})$

Postcondition: $Q(\bar{x}, \bar{y})$

$$\forall \bar{x}\ \exists \bar{y}\ P(\bar{x}) \rightarrow Q(\bar{x}, \bar{y})$$



# Recall: Levels of Formality



# Very High Level Formal Languages

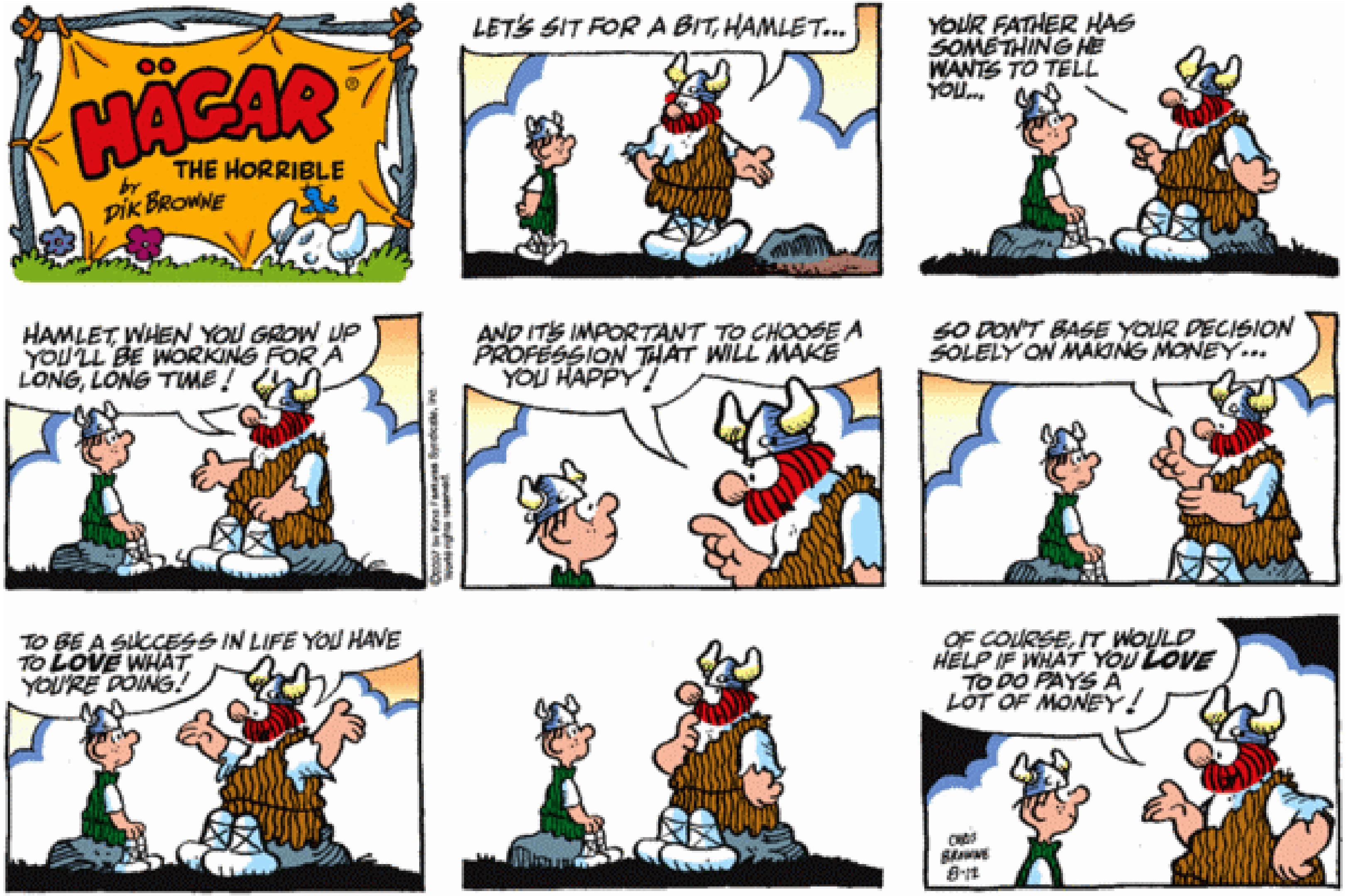
```
prev := {}; val := {};
val(x) := $\boxed{\{x\}}$ ;
(while newnodes $\neq$ {})
    n $\boxed{\text{from}}$ newnodes;
    ($\boxed{\forall m \in}$ graph {n})
        newval := val(n) + cost(n, m);
        if val(m) = $\boxed{\text{om}}$ or val(m) > newval then
            val(m) := newval;
            prev(m) := n;
            if m $\neq$ y then newnodes $\boxed{\text{with :=}}$ m; end if;
        end if;
    end $\forall$;
end while;
```

# Programming by Example

{( ) → ( )<br>
(*A B*) → (*A*)<br>
(*A B C D*) → (*A B*)<br>
(*A B C D E F*) → (*A B C*)}

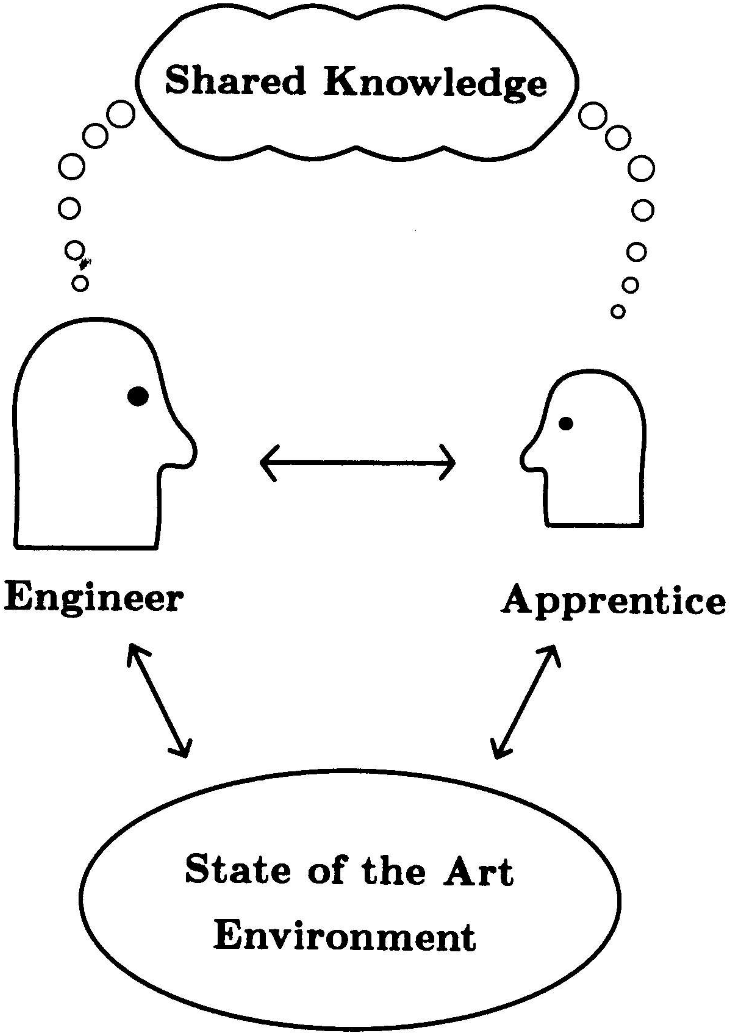
```
(DEFUN HALF (X)
   (H X X))

(DEFUN H (X Y)
   (COND ((ATOM Y) NIL)
         (T (CONS (CAR X)
                  (H (CDR X) (CDDR X))))))
```

# Computing Profession Choice… ☺


HÄGAR® THE HORRIBLE
by DIK BROWNE
LET'S SIT FOR A BIT, HAMLET…
YOUR FATHER HAS SOMETHING HE WANTS TO TELL YOU…
HAMLET, WHEN YOU GROW UP YOU'LL BE WORKING FOR A LONG, LONG TIME!
©2007 by King Features Syndicate, Inc. World rights reserved.
AND IT'S IMPORTANT TO CHOOSE A PROFESSION THAT WILL MAKE YOU HAPPY!
SO DON'T BASE YOUR DECISION SOLELY ON MAKING MONEY…
TO BE A SUCCESS IN LIFE YOU HAVE TO LOVE WHAT YOU'RE DOING!
OF COURSE, IT WOULD HELP IF WHAT YOU LOVE TO DO PAYS A LOT OF MONEY!
CHRIS BROWNE 8-12

# The Assistant Approach

- Productivity: Delegate routine details
- Reliability: Standardization of common practices



# Example: Table-Lookup

```
function table-lookup(table, key)
bucket ← table[hash(key)]
loop
    if bucket = nil then return nil
    entry ← head(bucket)
    if key(entry) = key then return entry
    bucket ← tail(bucket)
```

# Table-Insert

```
procedure *table-insert*(*table*, *entry*)
*push*(*entry*, *table*[*hash*(*key*(*entry*))])
```

# Table-Delete

```
procedure table-delete(table, key)
index ← hash(key)
bucket ← table[index]
if key(head(bucket)) = key
   then table[index] ← tail(bucket)
   else bucket-delete(bucket, key)
return table
```

# Bucket-Delete

```
procedure bucket-delete(bucket, key)
previous ← bucket
loop
   bucket ← tail(previous)
   if bucket = nil then return nil
   if key(head(bucket)) = key then
        tail(previous) ← tail(tail(previous))
        return nil
previous ← bucket
```

# Analysis: Table-Lookup

```
function table-lookup(table, key)
bucket ← table[hash(key)]
loop
    if bucket = nil then return nil
    entry ← head(bucket)
    if key(entry) = key then return entry
    bucket ← tail(bucket)
```

Linear Search

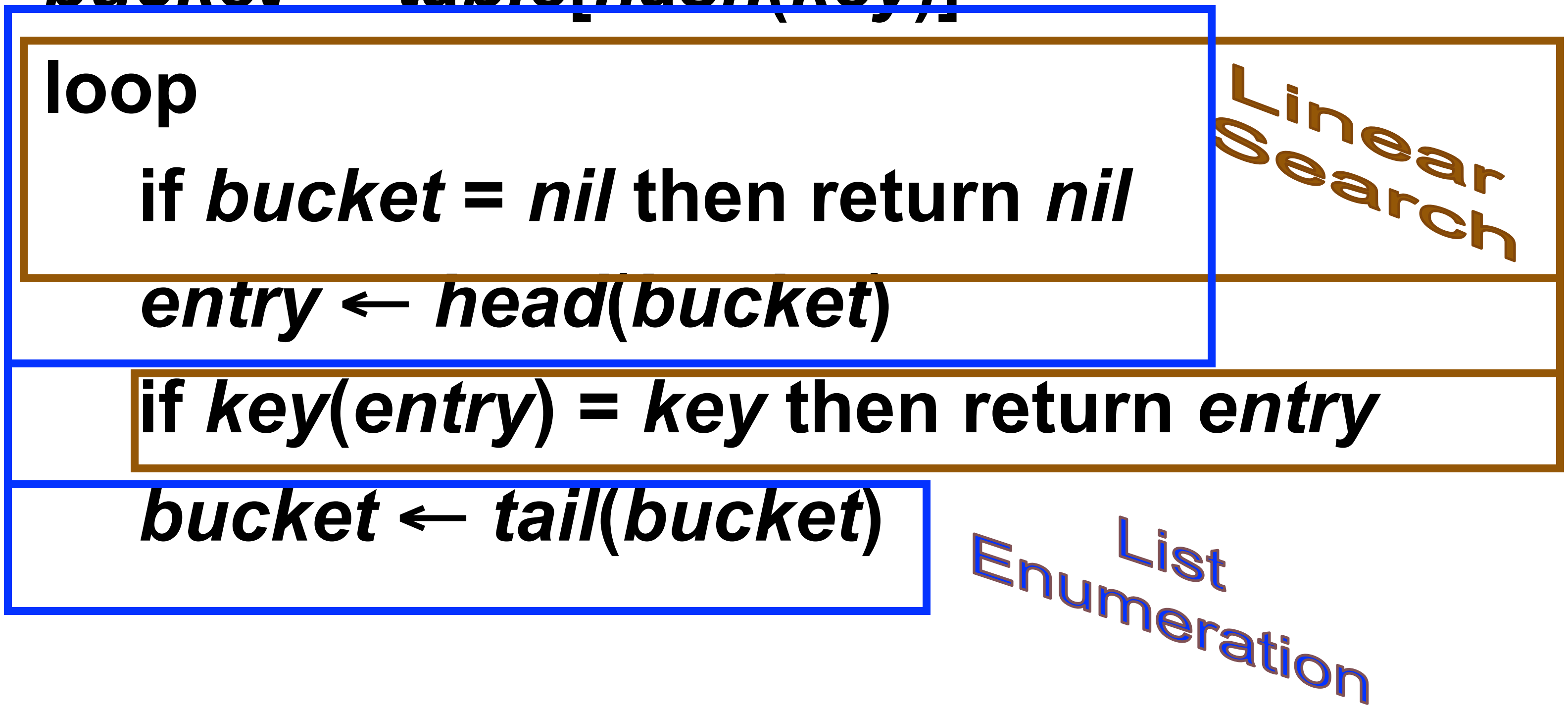
# Analysis: Table-Lookup

```
function table-lookup(table, key)
bucket ← table[hash(key)]
loop
    if bucket = nil then return nil
    entry ← head(bucket)
    if key(entry) = key then return entry
    bucket ← tail(bucket)
```

List Enumeration

# Analysis: Table-Lookup

```
function *table-lookup*(*table*, *key*)
*bucket* ← *table*[*hash*(*key*)]
loop
    if *bucket* = *nil* then return *nil*
    *entry* ← *head*(*bucket*)
    if *key*(*entry*) = *key* then return *entry*
    *bucket* ← *tail*(*bucket*)
```



# Analysis: Bucket-Delete

```
procedure bucket-delete(bucket, key)
previous ← bucket
loop
    bucket ← tail(previous)
    if bucket = nil then return nil
    if key(head(bucket)) = key then
        tail(previous) ← tail(tail(previous))
        return nil
previous ← bucket
```

Linear Search

# Analysis: Bucket-Delete

```
procedure bucket-delete(bucket, key)
previous ← bucket
loop
   bucket ← tail(previous)
   if bucket = nil then return nil
   if key(head(bucket)) = key then
         tail(previous) ← tail(tail(previous))
         return nil
previous ← bucket
```

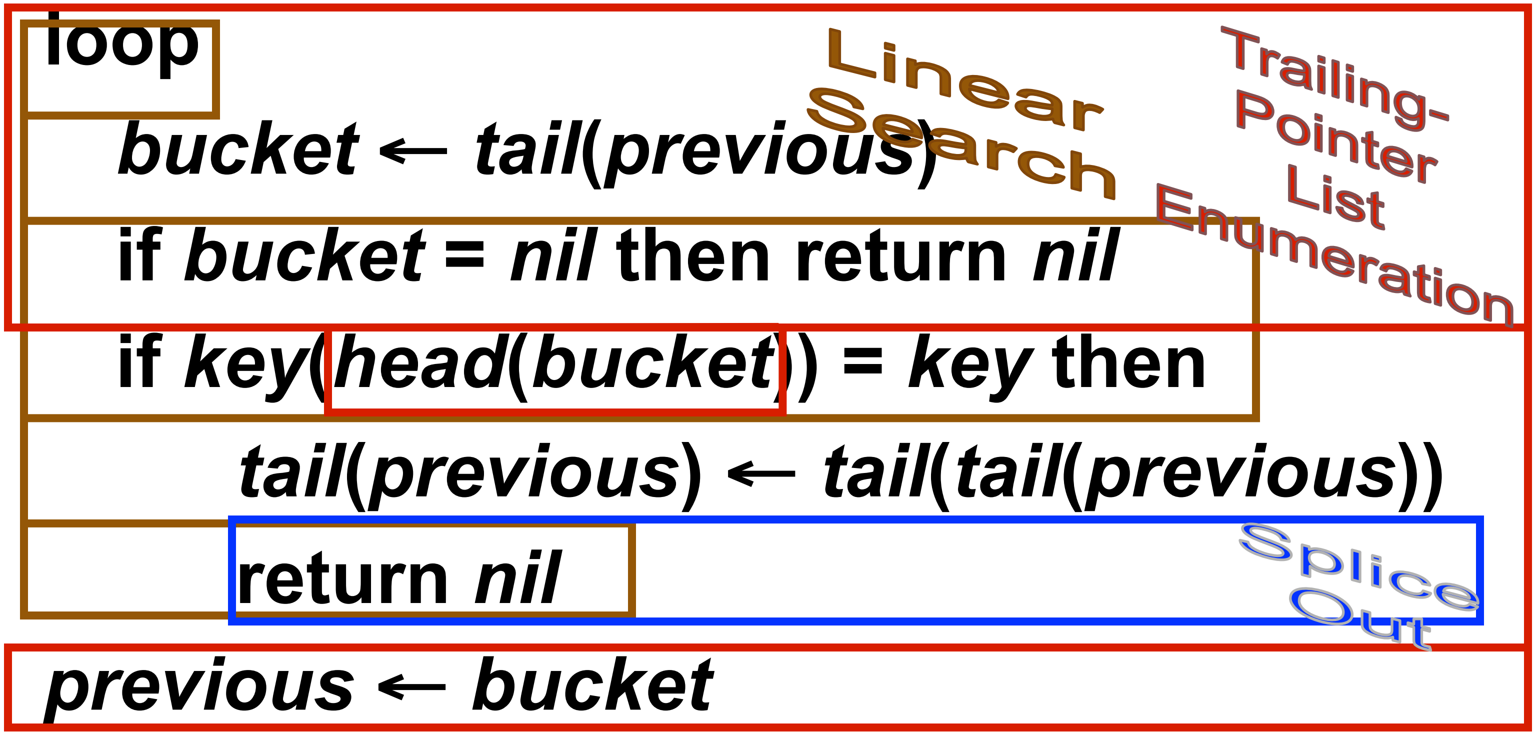
Trailing-Pointer List Enumeration

# Analysis: Bucket-Delete

```
procedure bucket-delete(bucket, key)
previous ← bucket
loop
    bucket ← tail(previous)
    if bucket = nil then return nil
    if key(head(bucket)) = key then
        tail(previous) ← tail(tail(previous))
        return nil
previous ← bucket
```

Splice Out

# Analysis: Bucket-Delete



```
procedure bucket-delete(bucket, key)
previous ← bucket
loop
    bucket ← tail(previous)
    if bucket = nil then return nil
    if key(head(bucket)) = key then
        tail(previous) ← tail(tail(previous))
        return nil
previous ← bucket
```

# Homework and Milestone Reminders

- Read Chapter 12 in text

- Milestone 3: Light-Weight Transformation Environment (see Milestone 3 assignment)
  - Due by 11:55pm, Thursday, May 5th, 2011.