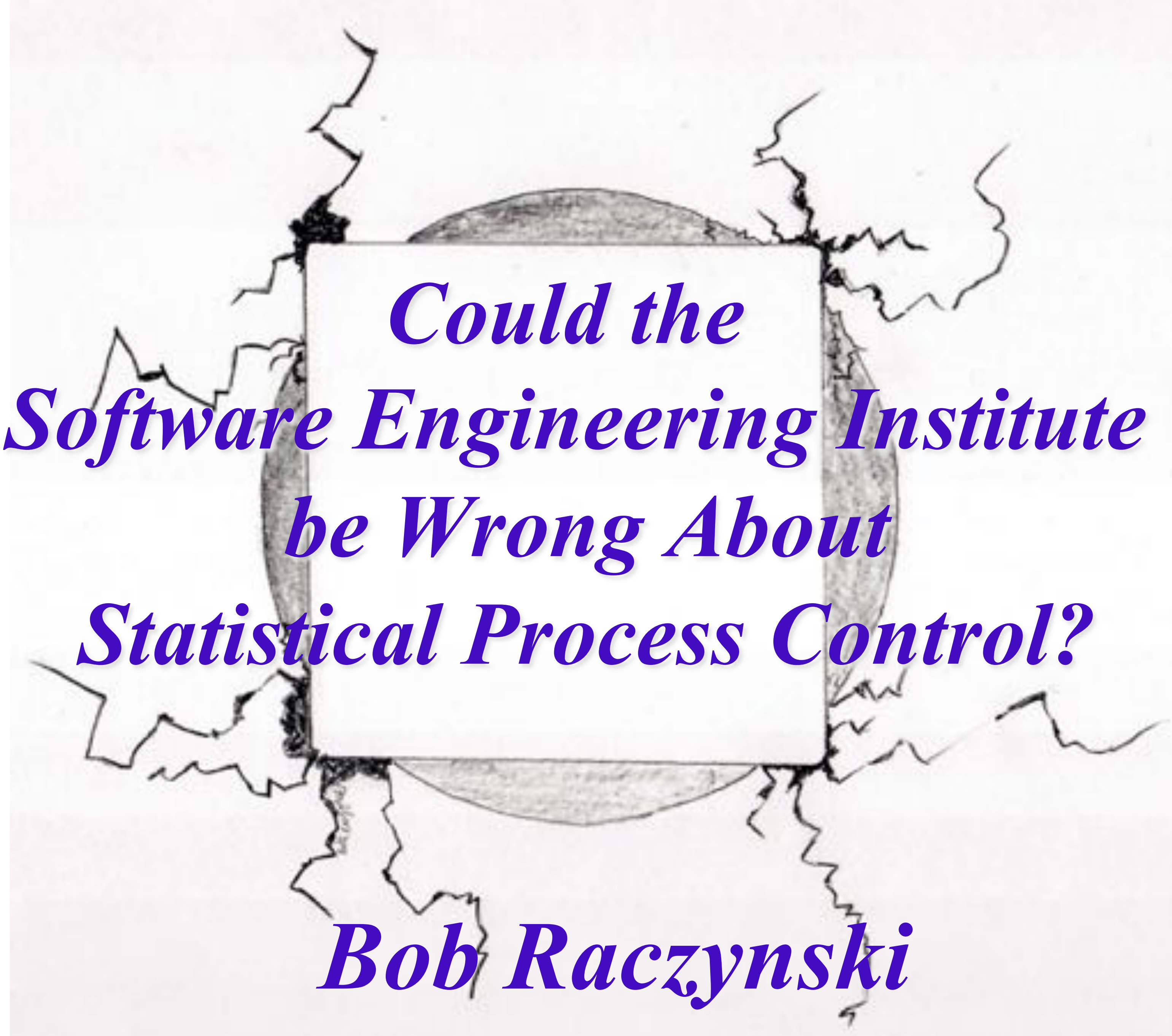
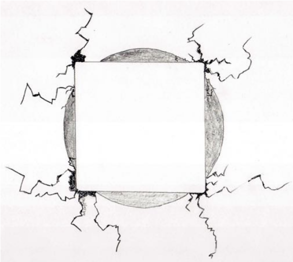


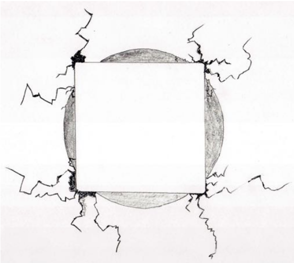
# Could the Software Engineering Institute be Wrong About Statistical Process Control?

Bob Raczynski



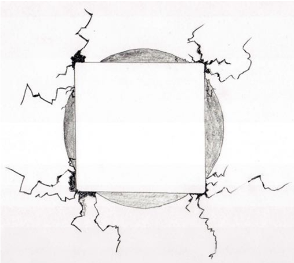
# *SPC – Some History*

- Invented by Dr. Walter A. Shewhart in the 1920s
- Popularized by Dr. W. E. Deming in the 1950s
- Transformed the manufacturing world
- The SEI Capability Maturity Model for Software (SW-CMM) included SPC as an integral component in the early 1990s - in the name of "predictability of process performance."



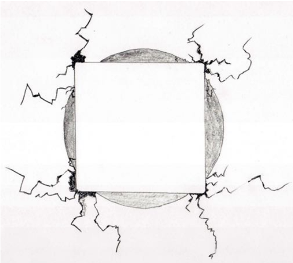
# *What is SPC?*

- "SPC is a way of thinking which happens to have some tools attached." – Dr. Donald J. Wheeler
- 2 main concepts:
  - Eliminate assignable (special) causes of variation where appropriate (uncontrolled variation)
  - Understand normal (common) causes of variation (chance variation)



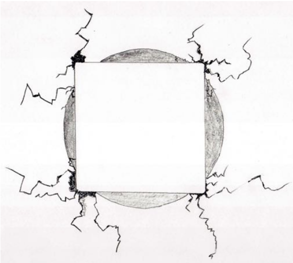
# *CMMI for Development, Version 1.2*

- Maturity Level 4: Quantitatively Managed
  - "Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences."
- Maturity Level 5: Optimizing
  - "Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes."



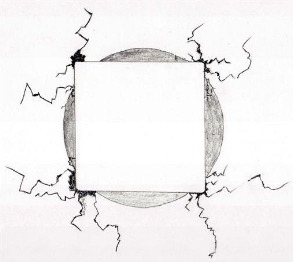
# CMMI for Development, Version 1.2

- The term "variation" is used 83 times in the CMMI.
- The term "special cause" is used 39 times in the CMMI.
- The term "common cause" is used 19 times in the CMMI.



# CMMI for Development, Version 1.2 – QPM PA

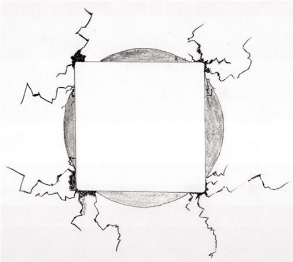
- SG 2 Statistically Manage Subprocess Performance
  - SP 2.1 Select Measures and Analytic Techniques
  - SP 2.2 Apply Statistical Methods to Understand Variation
  - SP 2.3 Monitor Performance of the Selected Subprocesses
  - SP 2.4 Record Statistical Management Data



# CMMI for Development, -Version 1.2

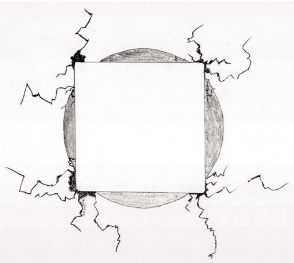
"**Statistically managed process –**

A process that is managed by a statistically based technique in which processes are analyzed, special causes of process variation are identified, and performance is contained within well-defined limits."

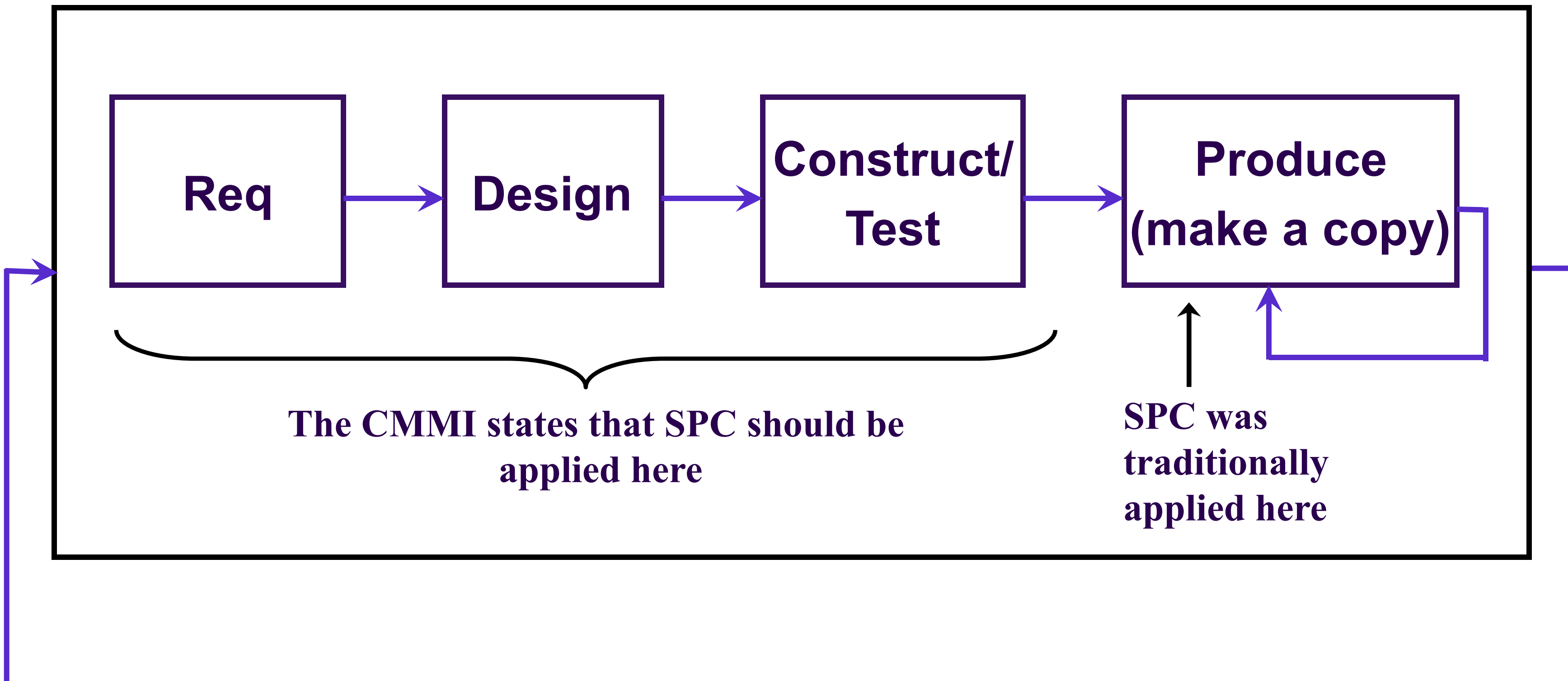


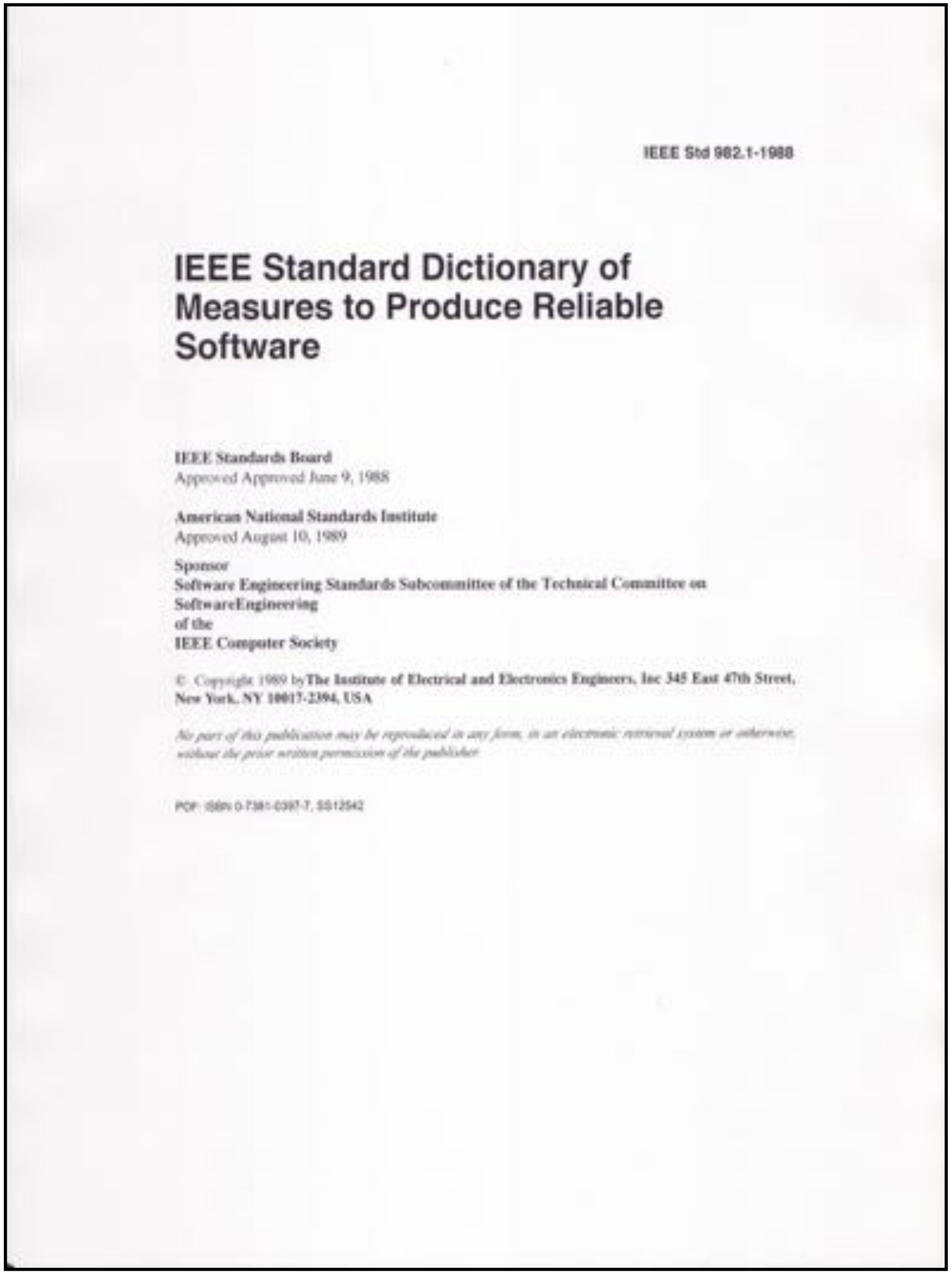
# *My Point:*

- SPC has a significant amount of emphasis placed upon it within the CMMI
- An organization can't exceed CMMI Level 3 without doing SPC



# Digital vs. Physical Product Manufacturing




IEEE Std 982.1-1988

IEEE Standard Dictionary of Measures to Produce Reliable Software

IEEE Standards Board
Approved Approved June 9, 1988

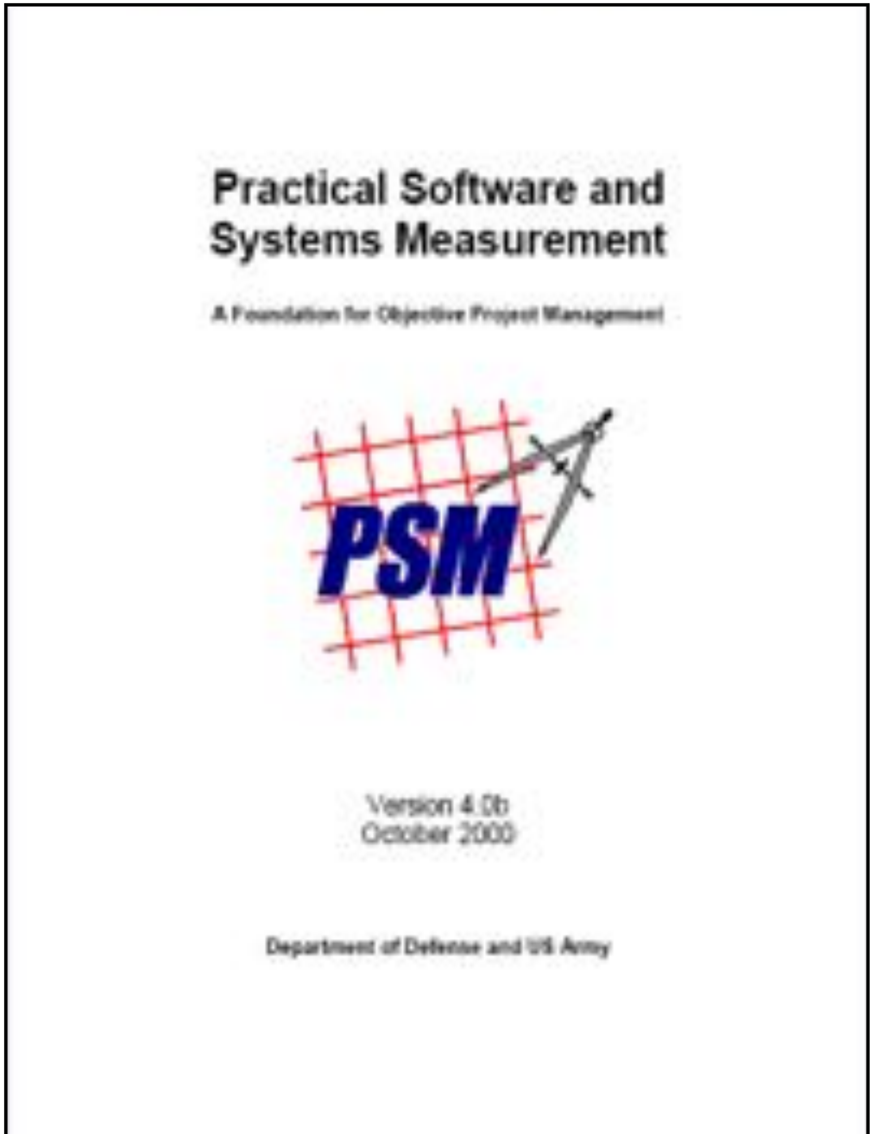
American National Standards Institute
Approved August 10, 1989

Sponsor
Software Engineering Standards Subcommittee of the Technical Committee on SoftwareEngineering
of the
IEEE Computer Society



PDF: ISBN 0-7381-0397-7, SS12542

# Contains no recommendation for SPC


Practical Software and Systems Measurement

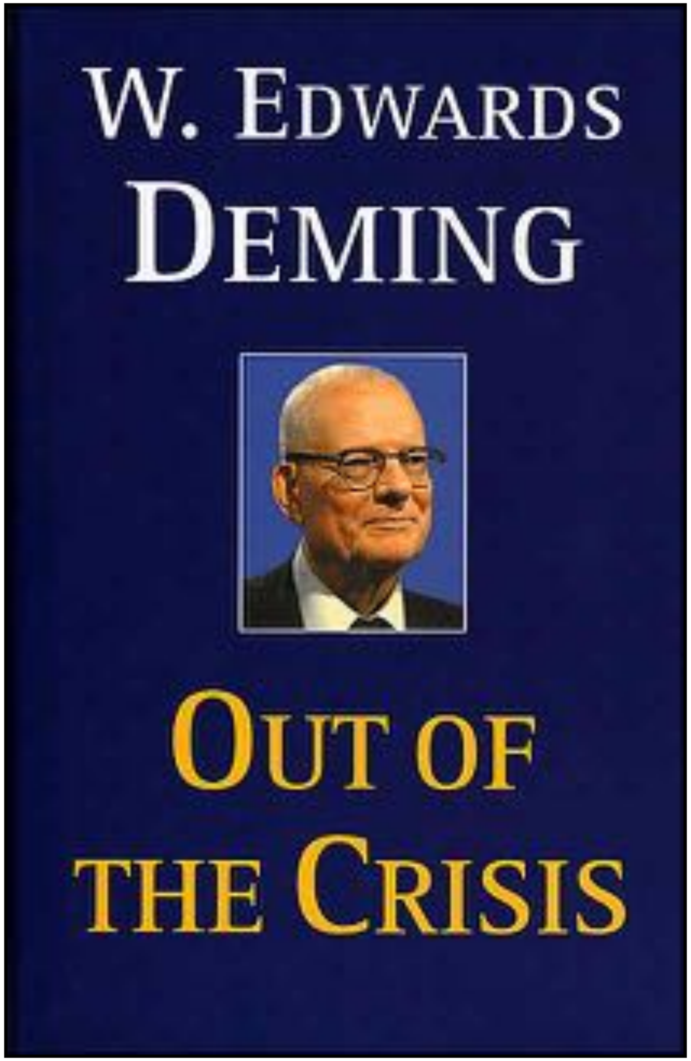
A Foundation for Objective Project Management

PSM

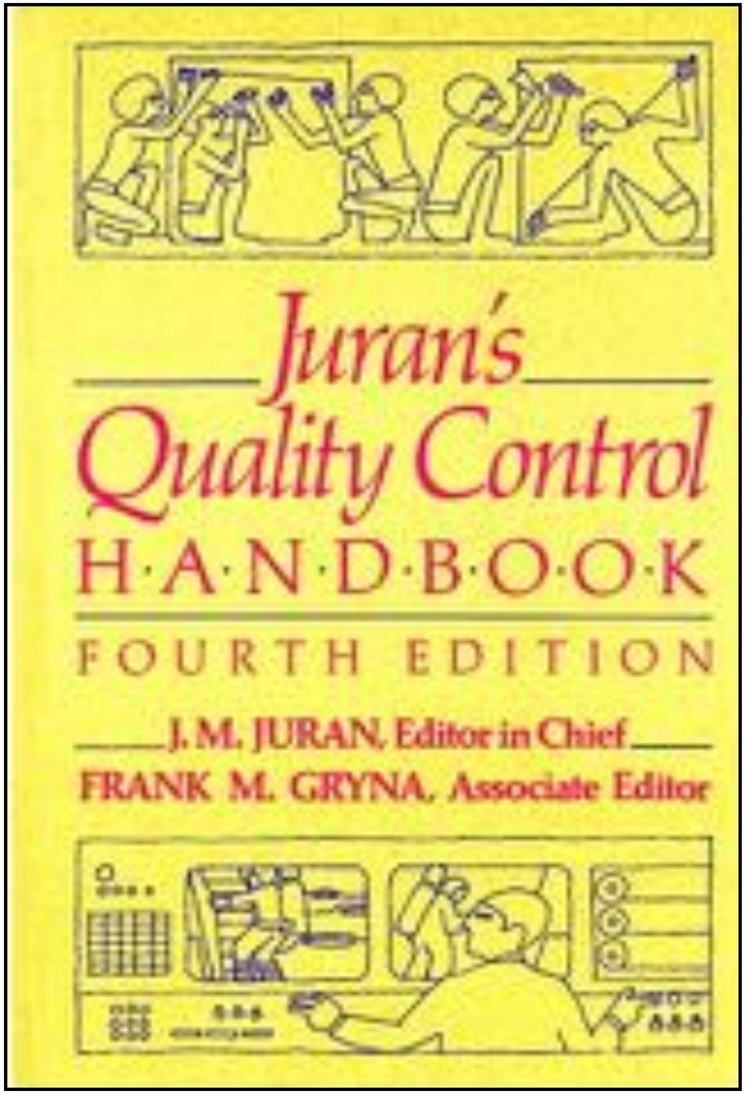
Version 4.0b
October 2000

Department of Defense and US Army

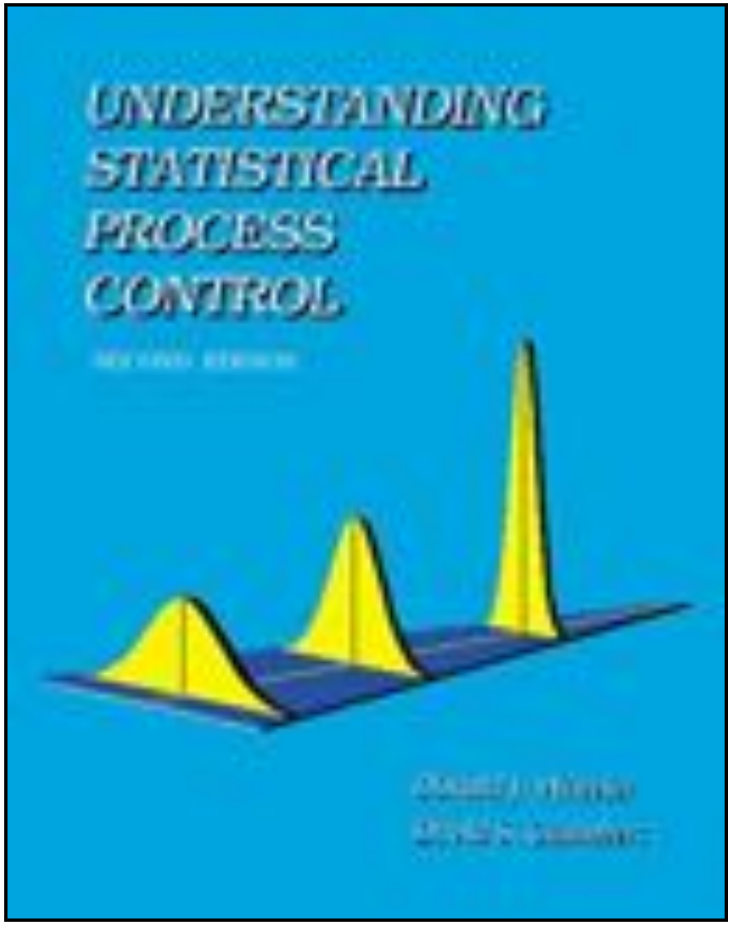
# Contains no recommendation for SPC



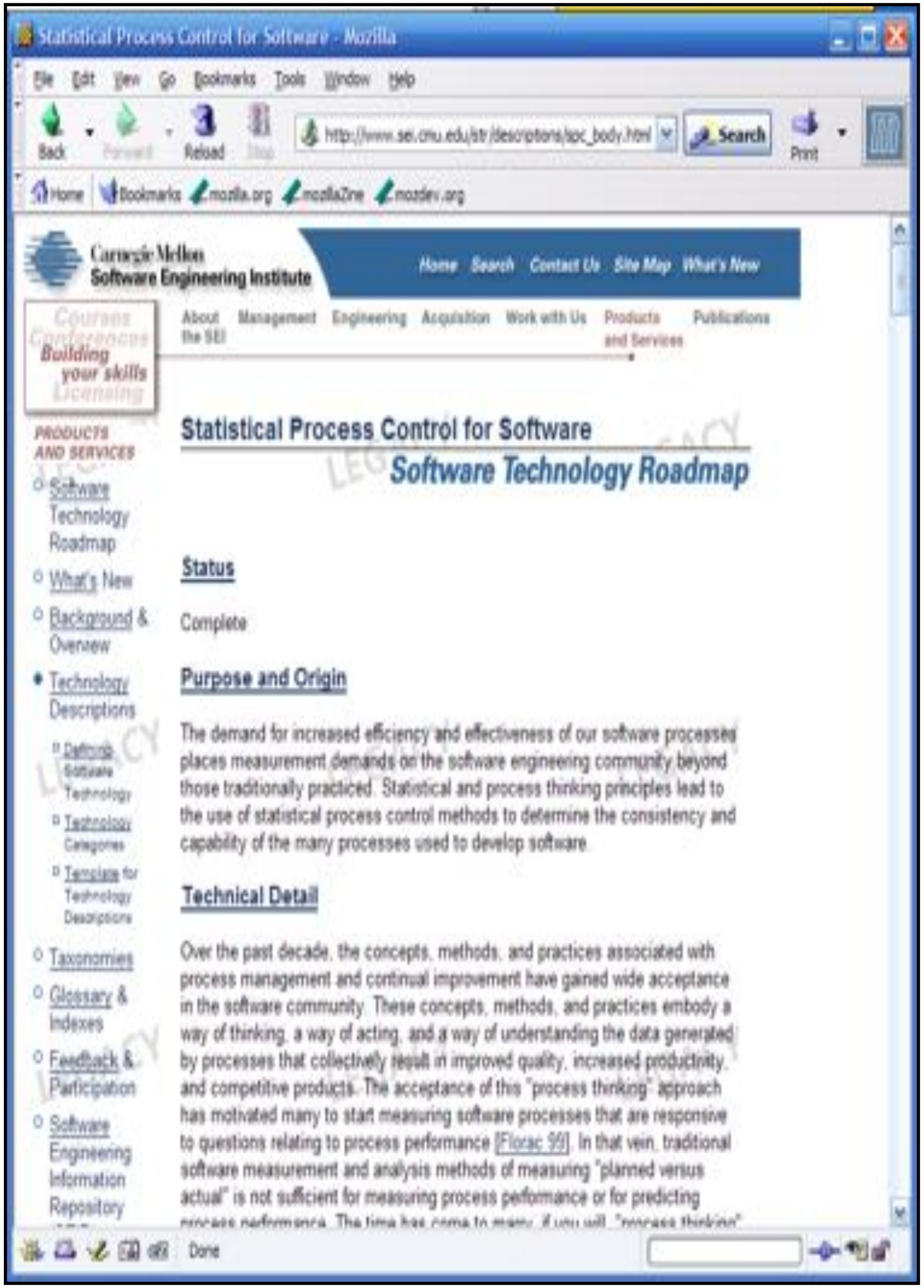

"3. Cease dependence on mass inspection. ... We must note that there are exceptions, circumstances in which mistakes and duds are inevitable but intolerable."



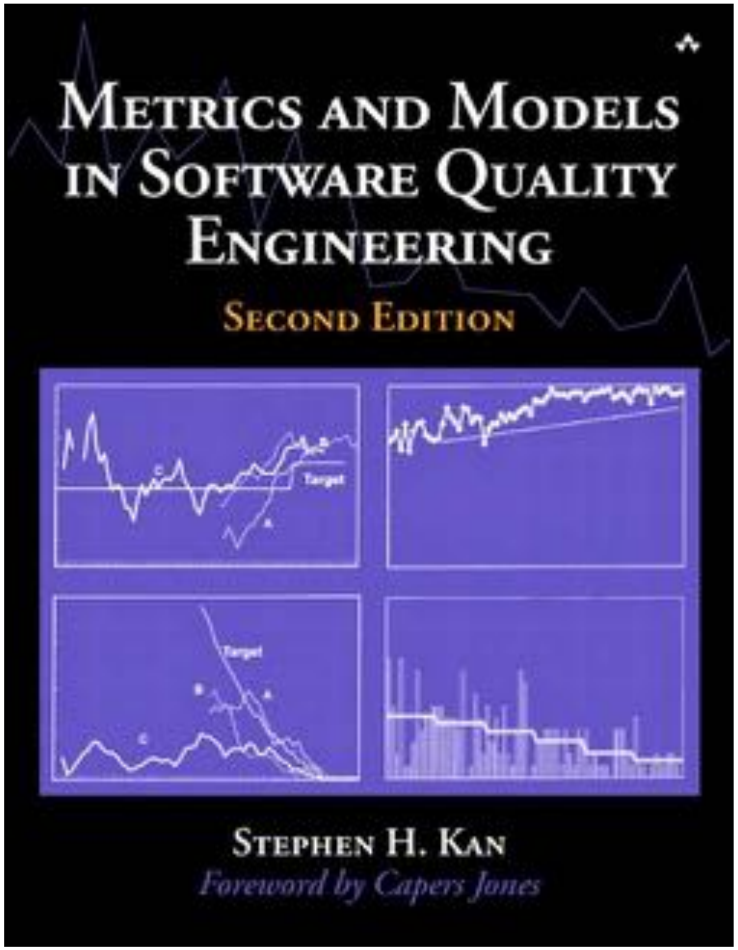

"Unfortunately, as is often the case in such matters, Shewhart's prospectus has become orthodoxy for many of today's quality control practitioners."



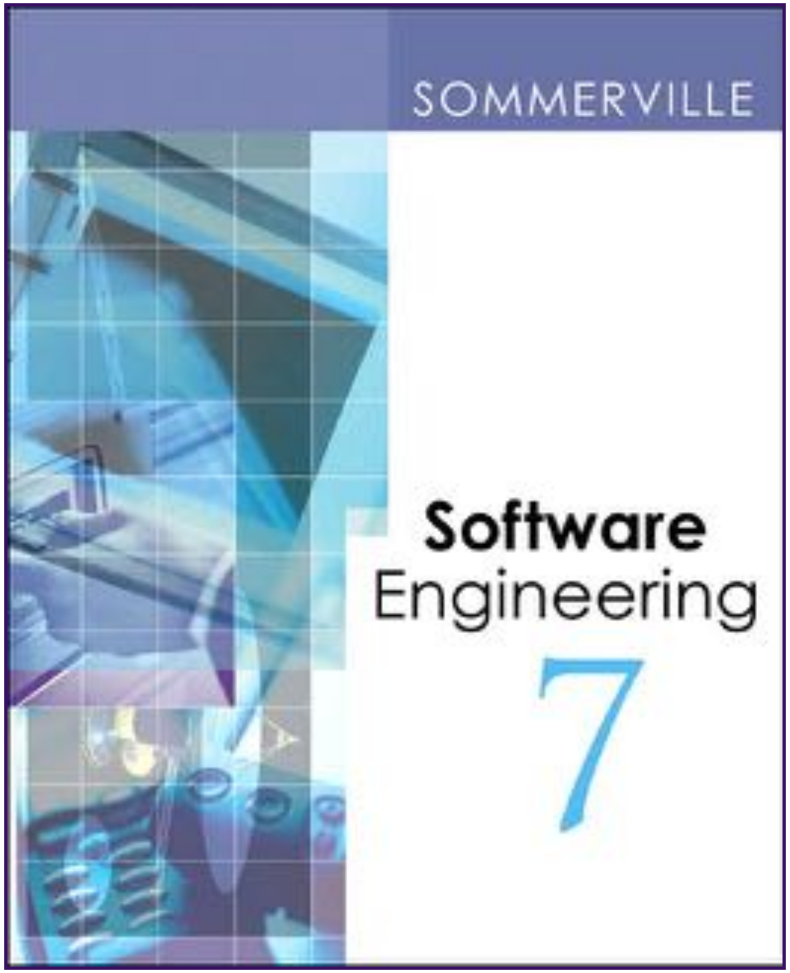

"Attribute Data differ from Measurement Data in two ways. First of all Attribute Data have certain irreducible discreteness which Measurement Data do not possess. Secondly, every count must have a known 'Area of Opportunity' to be well-defined"




"Control limits become wider and control charts less sensitive to assignable causes when containing non-homogeneous data"




"However, in software development it is difficult to use control charts in the formal SPC manner. It is a formidable task, if not impossible, to define the process capability of a software development process"




**Quote on next page**

First Sommerville quotes Watts Humphrey:

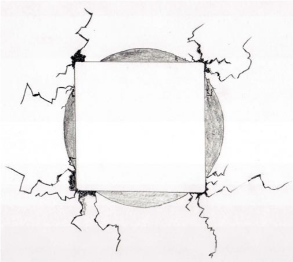
"W. E. Deming, in his work with the Japanese industry after World War II, applied the concepts of statistical process control to industry. While there are important differences, these concepts are just as applicable to software as they are to automobiles, cameras, wristwatches and steel."

Sommerville then goes on to state:

"While there are clearly similarities, I do not agree with Humphrey that results from manufacturing engineering can be transferred directly to software engineering. Where manufacturing is involved, the process/product relationship is very obvious. Improving a process so that defects are avoided will lead to better products. This link is less obvious when the product is intangible and dependent, to some extent, on intellectual processes that cannot be automated. Software quality is not dependent on a manufacturing process but on a design process where individual human capabilities are significant."

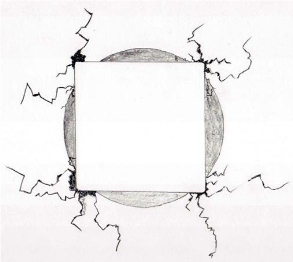

# Doesn't prescribe SPC


CERTAIN DEATH BEYOND THIS POINT!
DANGER VERY THIN ICE.
THIN ICE



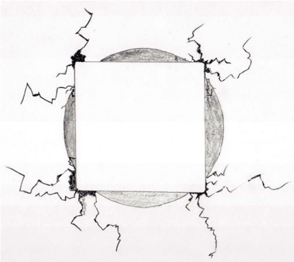
# *Problems*

- The following slides present four specific problems which one faces when attempting to apply SPC to a human-intensive, knowledge-intensive process



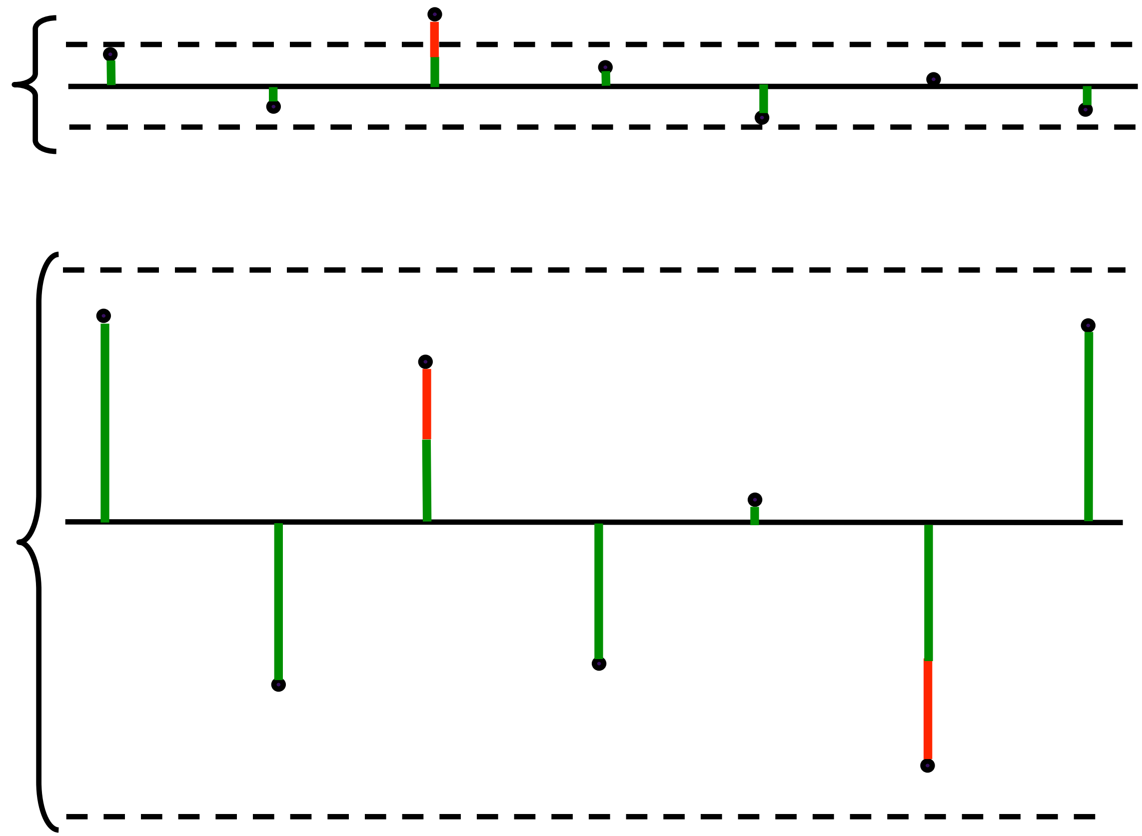
# *Problem #1 – Wide Control Limits*

- When the normal variation is great (as in human-intensive, knowledge intensive processes) the control limits of the control charts become very wide, and almost all variation is considered normal



# *Problem #1 - Wide Control Limits*

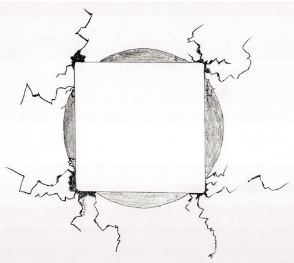
Narrow control limits



Wide control limits

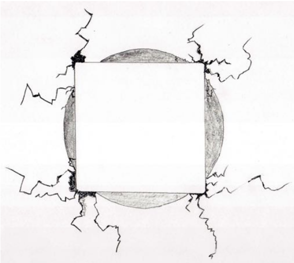
Variation resulting from normal causes

Variation resulting from an abnormal event



# *Problem #2 – Impossible to eliminate all assignable causes*

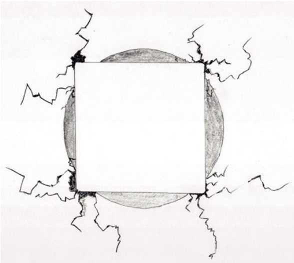
- First you have to detect them
- Then you have to identify them



# *Problem #2 – Impossible to eliminate all assignable causes*

Some possible causes of variation in a human-intensive process:

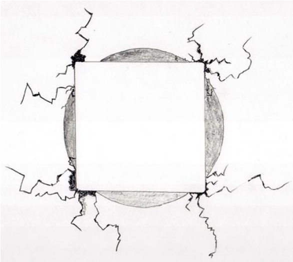
- Different people
- Same people, but one or more of the following applies to one or more of the people:
  - Has more job stress
  - Doesn't feel well
  - Has more family stress
  - Just quit smoking
- Lack of sleep
- Not enough caffeine
- Going through a divorce
- Mom died
- Lack of nutrition
- Under a schedule crunch
- In a bad mood



# Problem #2 – Impossible to eliminate all assignable causes

## Some more possible causes of variation in a human-intensive process:

- Bitter due to lack of recognition
- Has a cold
- Distracted due to automobile issues
- Lack of exercise
- Distracted due to political issues
- Is cold
- Is hot
- Being bothered by mother-in-law
- Has health issues
- Is becoming unsatisfied with job
- Has a toothache
- Is tired
- Is not familiar with the piece of code being inspected
- Is hung-over
- Found out that he/she needs surgery
- Is recovering from surgery
- Is feeling depressed

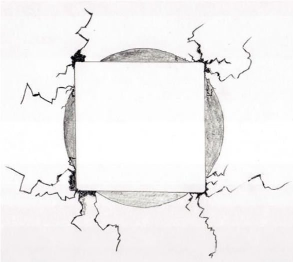


# *Problem #2 – Impossible to eliminate all assignable causes*

- The list goes on and on

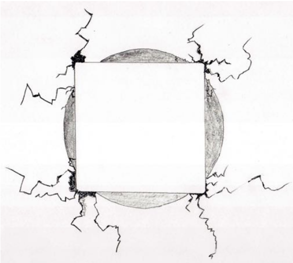
My point:

- In any human-intensive, knowledge-intensive process, assignable causes that are detected:
  - Are difficult if not impossible to identify and
  - Even if identified, are difficult if not impossible to eliminate from the process (much easier with machines)



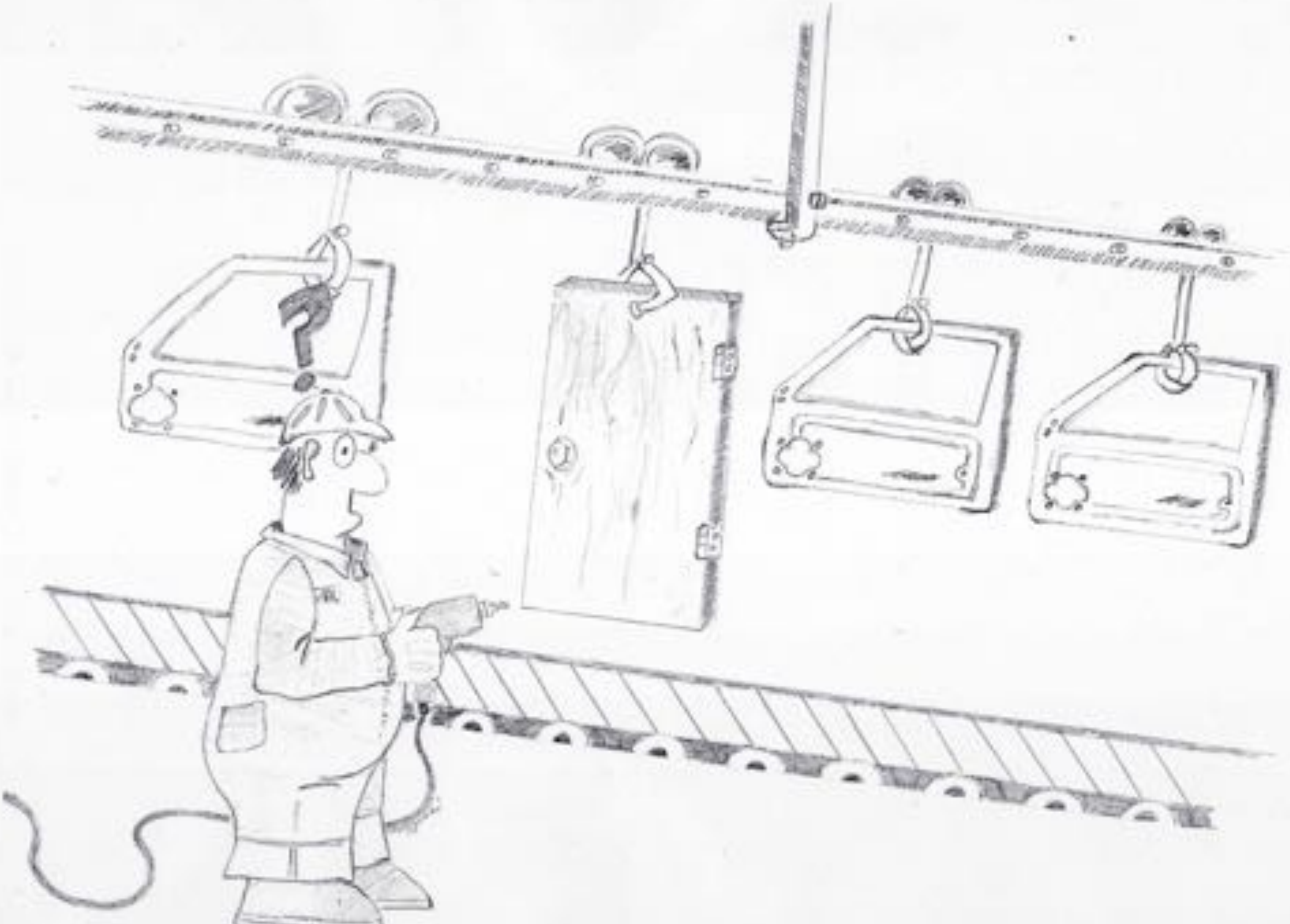
# *Problem #3 – Each Individual Process is Different From Invocation to Invocation*

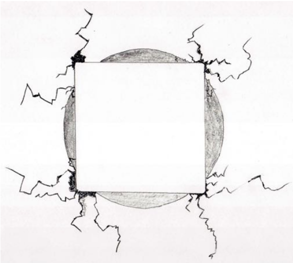
- No statistician alive would ever mix data from different assembly lines in a single control chart
- Yet, that is exactly what happens when people attempt to apply SPC to software development process



# *Problem #3 - Each Individual Process is Different From Invocation to Invocation -Are all processes alike?*

| | Processing elements between invocations virtually identical | Processing elements between invocations are different, but are in the same class | Processing between invocations in different class |
|---|---|---|---|
| **Inputs between invocations virtually identical** | **Manufacturing process** | | |
| **Inputs between invocations are different, but are in the same class** | | **Software Inspection** | |
| **Inputs between invocations in different class** | | | |

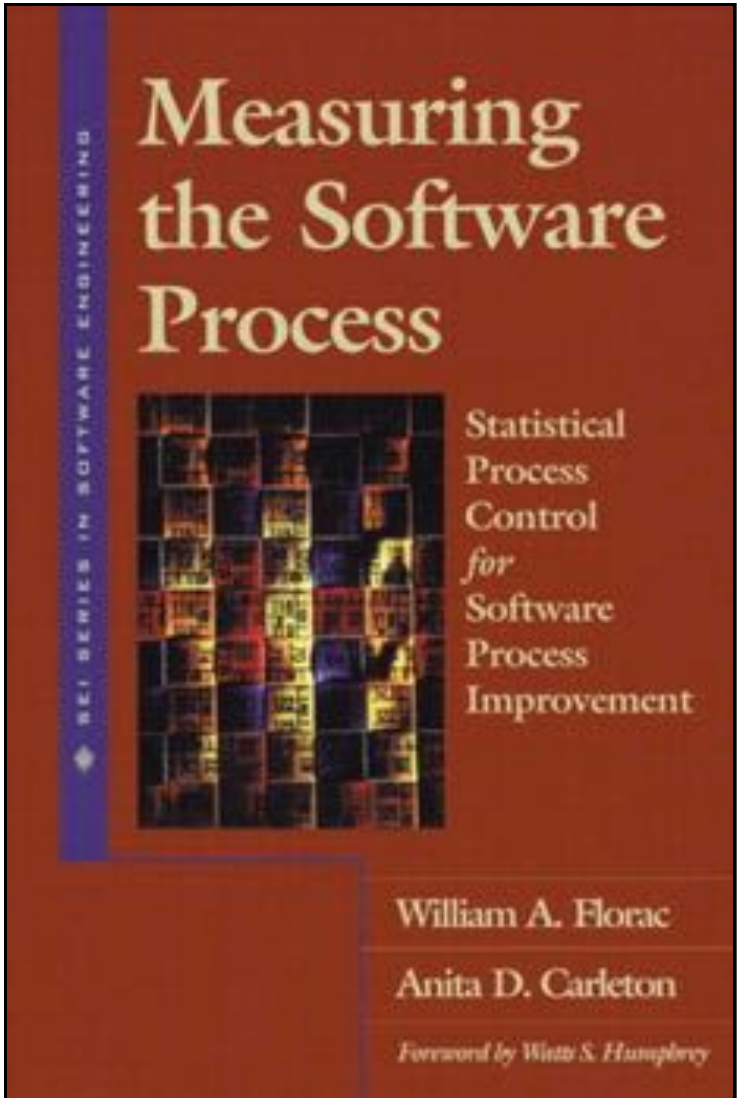




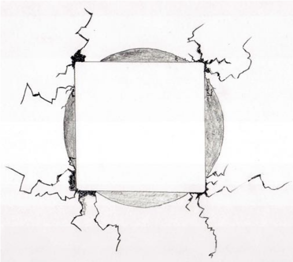
# *Problem #3 – Each Individual Process is Different From Invocation to Invocation*

- With every invocation of a software development process:
  - The input(s) to the process are not virtually identical
  - The processing elements are not virtually identical
- In other words, there are multiple common cause systems present which are difficult if not impossible to isolate (resulting in non-homogeneous data)
- This is a fundamental distinction between manufacturing processes and human-intensive, knowledge-intensive processes

# *Problem #4 - Can't normalize the data*




"We can conceive of situations, such as variations in the complexity of internal logic or in the ratios of executable to nonexecutable statements, where simply dividing by module size provides inadequate normalization to account for unequal areas of opportunity."



# *Problem #4*
# *Can't normalize the data*

- The area of opportunity is not easily quantifiable, therefore normalizing the data isn't practical

- The code samples on the following two slides have vastly different areas of opportunity

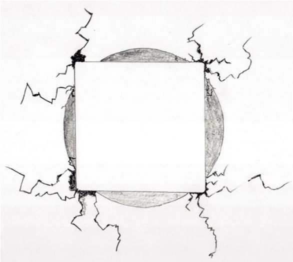
McCabe Cyclomatic Complexity = 4
Number of Logical SLOC = 31

```
#include <stdio.h>
#include <strings.h>
#include <stdlib.h>

int main(void)
{
 int number1, number2, number3, number4, number5, number6, ii;

 printf("\nPlease enter a number: "); scanf("%d", &number1);
 printf("\nPlease enter another number: "); scanf("%d", &number2);

 if ( number1 > number2 ) {
  printf("\nThe first number is greater");
 }
 if ( number2 > number1 ) {
  printf("\nThe second number is greater");
 }
 if ( number1 == number2 ) {
  printf("\nThe first and second numbers are equal");
 }

 number3 = number1 + number2; printf("\nAddition:  %d + %d = %d", number1, number2, number3);
 number4 = number1 - number2; printf("\nSubtraction:  %d - %d = %d", number1, number2, number4);
 number5 = number1 * number2; printf("\nMultiplication:  %d x %d = %d", number1, number2, number5);
 number6 = number1 / number2; printf("\nDivision:  %d / %d = %d (no remainder calculated)", number1, number2, number6);

 printf("\n\n Have a nice day!");
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
void main( void )
{
   #define VAL 63
   struct d {
      short a,b,c;
      float aa;} e;
   register int xx = -12;
   volatile char  cf = ~xx;
   short fc = cf<<1;
   memset( &e,0, sizeof e);
   xx = scanf( "%d %c", &fc, &cf );
   if(xx=cf=='f'?1:0) {
      e.a = 5;
      e.b = (e.a<<1)-1;
      e.c = (0x10)<<(2%e.a>>1);
      e.aa = ((((float)(e.a))*(float)(e.b<<1))/(e.b*(VAL^45)))*(float)(fc-( e.c == 1+(VAL>>1) ? 32:-44));
      printf( "%c = %f\n", 'A'+2, e.aa );}
   else {
      short a[VAL&~060] = { (VAL>>3)-2 };
      short * ptr = &a[ ((sizeof a)>>1) & 0xfff5 ], **pptr = &ptr, *pttr = a;
      float * eaptr = &e.aa;
      *(++pttr) = e.c + 2*a[0] - e.a - (VAL>>a[0]);
                  *(ptr -= 3) = ((*pttr + a[0])<<1) + sizeof (int);
      *eaptr = (float)(**pptr) + ((float)(*pttr*fc))/(float)a[0];
      printf("\nF = %f\n", e.aa );}
}
```

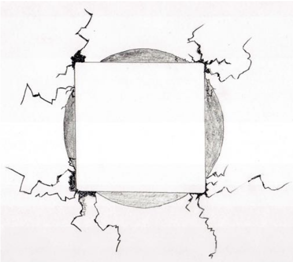
McCabe Cyclomatic Complexity = 4

Number of Logical SLOC = 31



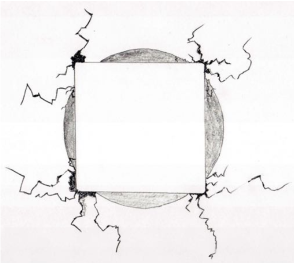
# *Problem #4*
# *Can't normalize the data*

- What is the area of opportunity of the previous two code samples?
  - I don't know
  - I have no way to accurately quantify it
  - I can subjectively state that the latter code sample has a far greater area of opportunity than the prior due to the complexity of the code



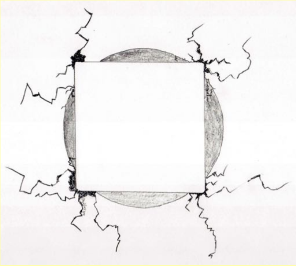
# *Can we get around these problems?*

| Problem | Typical "Solution(s)" |
| --- | --- |
| Problem #1 – Wide control limits | Assert that SPC is still applicable, the control limits are just wide |
| Problem #2 – Impossible to eliminate all assignable causes | Pretend that assignable causes are just like those present in manufacturing processes (can be easily detected, identified, and removed) |
| Problem #3 – Each Individual Process is Different From invocation to invocation | Assert that all processes are equal and continue to advocate SPC as a silver bullet. (Software Engineering = Hardware Manufacturing) |
| Problem #4 - Can't normalize the data | Divide by logical SLOC, separate data-lists, tables, and arrays from other code. Then claim that the unequal areas of opportunity have been accounted for. |



# *Can we get around these problems?*

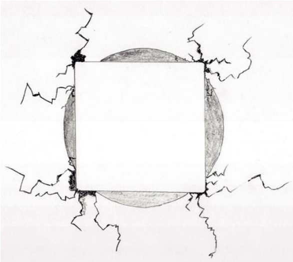
- Maybe the answer isn't to try so hard to get around the problems.
- Maybe SPC just isn't the right tool.
- Maybe we are trying too hard to fit a square peg into a round hole.



# *So was the SEI wrong about SPC for SW Dev processes?*

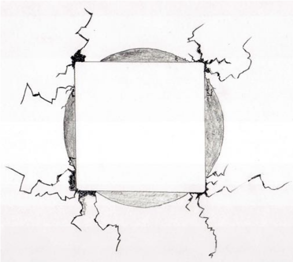
- Well, if you do apply SPC to software development processes, you might occasionally get lucky and **detect** an assignable cause of variation despite the wide control limits of your control charts, unequal area of opportunities, and ever-changing processes
- Of the assignable causes of variation that you do manage to detect, you might occasionally get lucky and actually **identify** one of the causes of that variation
- Of the very few assignable causes of variation that you manage to identify, one of them might occasionally be of a nature in which it can actually be **removed** with some persistence
- Even so, your overall system will hardly be more **predictable**

**Is this the best use of your limited resources?**



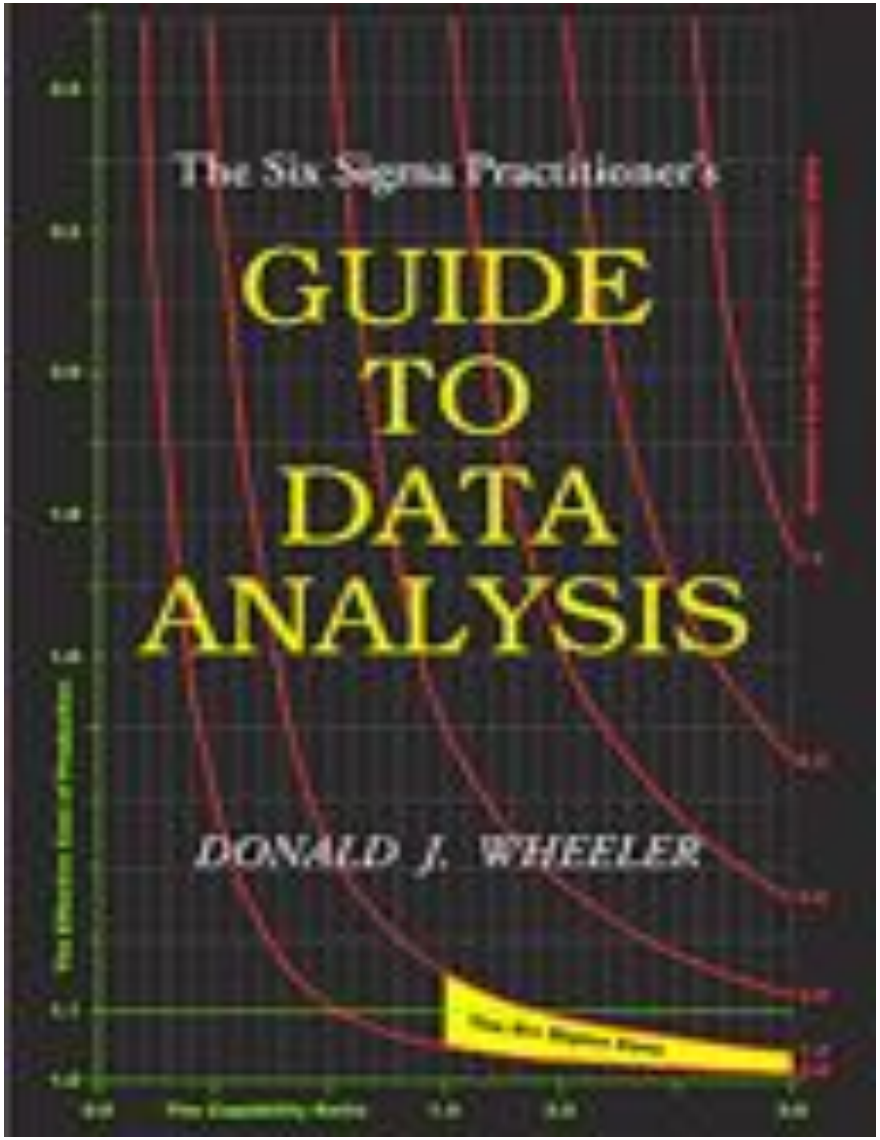
# *So was the SEI wrong about SPC?*

- Technically, one can apply SPC to any process
- The Question is how useful doing so will be
- What the SEI got wrong is the amount of emphasis that they placed on using SPC with engineering processes

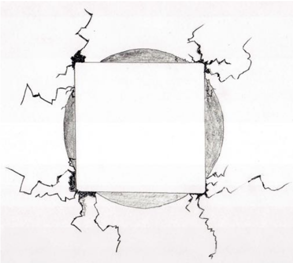


# *The other thing the SEI got wrong about SPC:*

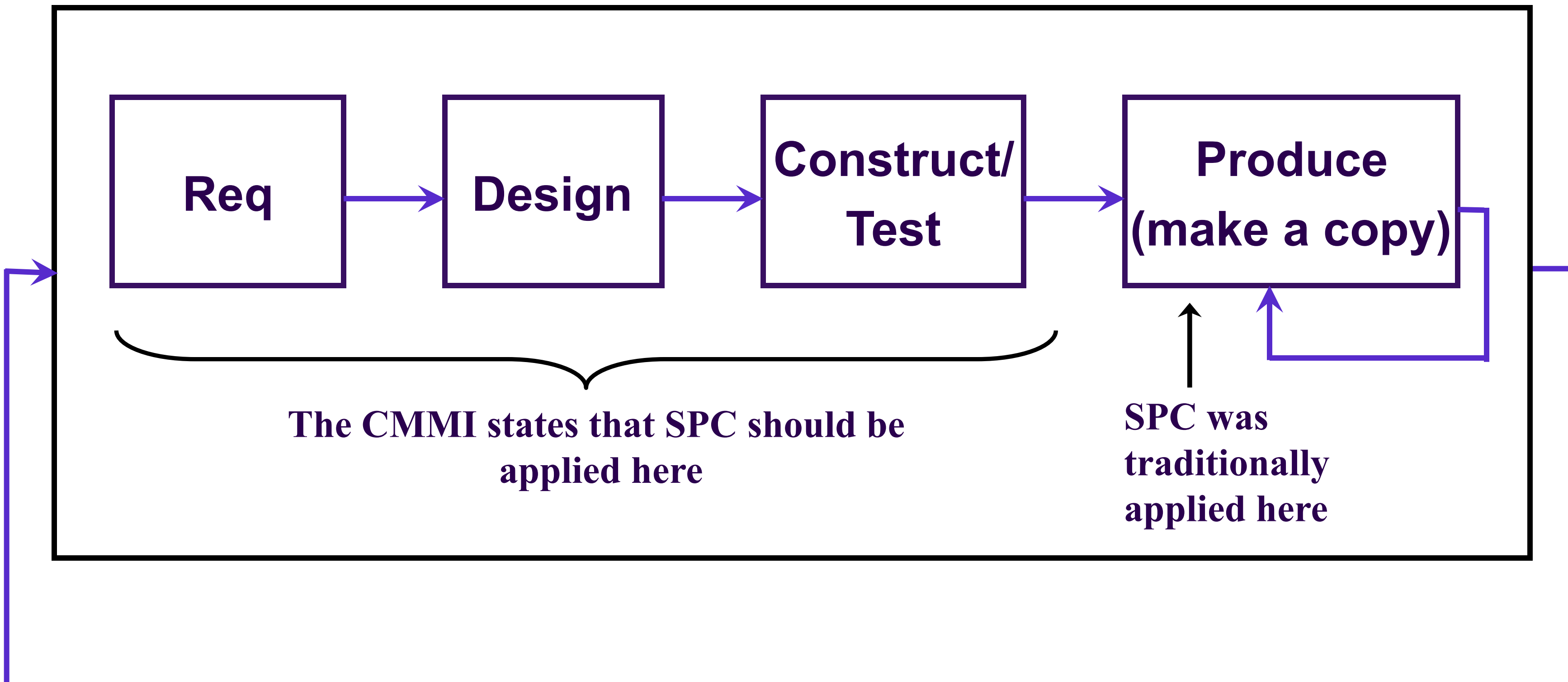
- Maturity Level 5: Optimizing
  - "Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes."

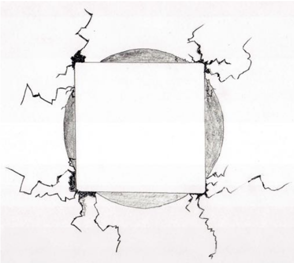



**"Since reengineering a process is never cheap, it should be undertaken only when it is needed."**

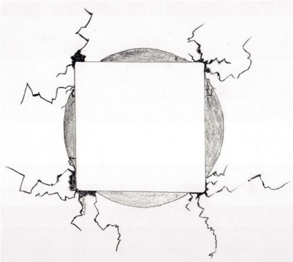


# *How did SPC get into the CMMI in the first place?*



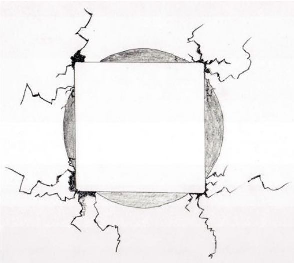


# *How did SPC stay in the CMMI for so long?*



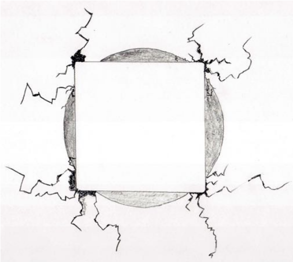
# *So, what is an alternative to SPC?*

- Have experienced and technical SQE on your staff.
- By working directly with the people while the development is progressing they can:
  - Recognize when abnormal events are happening (even without control charts)
  - Frequently prevent problems before they occur
  - My experience is that other quantitative techniques are more useful (e.g. inspection coverage report)
  - Lead process improvement efforts (even without control charts)
- For predictability of the overall system, use a parametric modeling tool (e.g. SLIM by QSM)



# *What the SEI needs to do*

- De-emphasize SPC
  - Recommend taking the same approach as ISO 9001:2000:
    "… shall include determination of applicable methods, including statistical techniques, and the extent of their use."
- Consider collapsing at least level 5, and possibly levels 4 and 5, as the distinctions are largely based on SPC



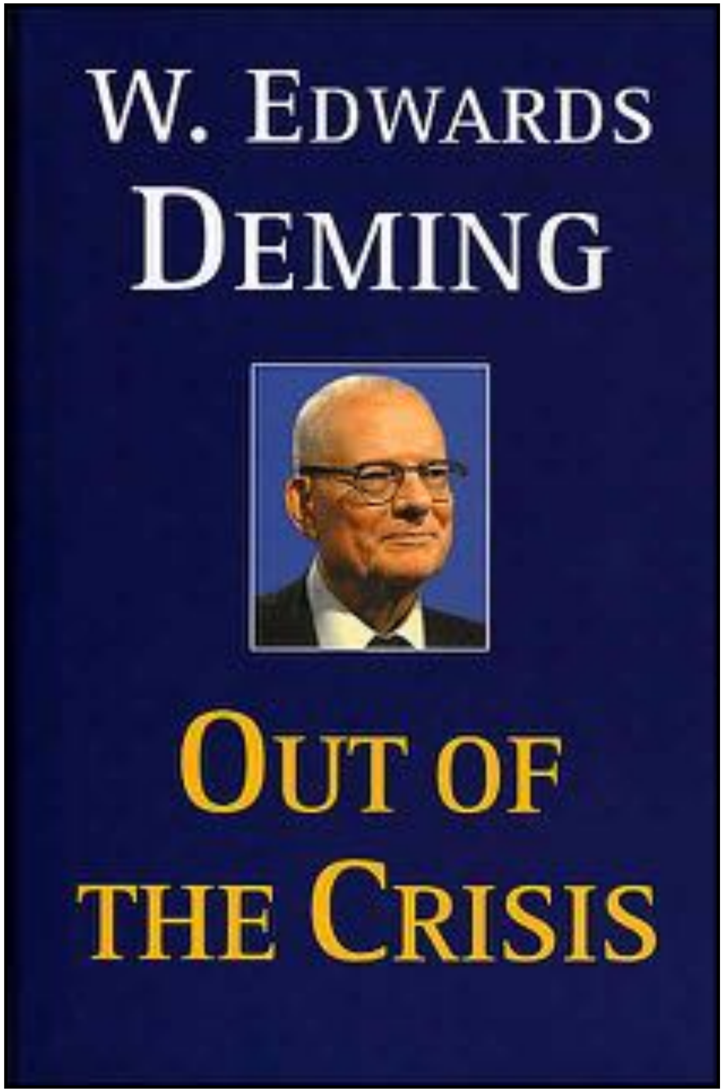
# *Director of the SEI - Nice guy. Give him a call*

- Paul Nielson
- nielsen@sei.cmu.edu
- +1 412 268 7740

# *My contact information*

- Bob Raczynski
- bobraczynski@computer.org
- +1 303 971 3907

# *One last Deming quote:*




**"It is not necessary to change. Survival is not mandatory."**

**- W. Edwards Deming**

# Acronym List

- PA – Process Area
- PSM – Practical Software and Systems Measurement
- QPM – Quantitative Project Management
- QSM – Quantitative Software Management
- SEI – Software Engineering Institute
- SG – Specific Goal
- SLIM – Software LIfecycle Management
- SLOC – Software Lines Of Code
- SP – Specific Practice
- SPC – Statistical Process Control
- SQE – Software Quality Engineer
- SW-CMM – Software Capability Maturity Model