TRAPPING & MASS PRODUCING KNOWLEDGE

Author: Jeffrey M. Setterholm
Lakeville, Minnesota
(~~612~~) 461-3445
952

Appendix A: An example of trapped technical knowledge - added March 11, 1993

I place this document & Appendix A in the
PUBLIC DOMAIN
January 20th, 2017

The last page has relevant links.

TRAPPING & MASS PRODUCING KNOWLEDGE
Initial Emphasis: Technology

Jeffrey M. Setterholm
Lakeville, Minnesota

INTRODUCTION

The Greater Minnesota Corporation seeks to develop and implement a blueprint for regional vitality. People raised in the region perceive improving education as an integral part of that vitality. Thus, Education is an area already under scrutiny.

A major goal of education is to increase the student's ability to apply knowledge: to solve problems, to understand phenomena, and to achieve desired objectives. But in order to teach or apply knowledge efficiently, first it needs to be available... efficiently available.

This paper explores a new strategy for improving education. Potential impact of this strategy on regional vitality, a complex issue, is also addressed.

Four claims:

1) **Enormous amounts of knowledge are being lost or re-created from one generation to the next.**

2) **As a WASTE-SAVING measure, the systematic TRAPPING of knowledge into the public domain deserves substantial support.**

3) **Education can be well-served by the knowledge trapping activity through the MASS PRODUCTION of KNOWLEDGE.**

4) **Regional economic vitality can be well served through MASS PRODUCTION of the HARDWARE and software that simplifies interaction with the knowledge.**

While these claims apply to most areas of knowledge, this paper focuses on technical knowledge because I understand how to efficiently trap it.

APPROACH OVERVIEW

The capabilities of unaided human memory are quite unpredictable and vastly overrated by both academia and the popular press. Computers, on the other hand, are extremely predictable and vastly underrated. Hence, a partnership of mind and computer is proposed… with the actual knowledge storage occurring in the reliable, exercisable, tireless, patient, objective, replicative, explicit, time-invariant, observable,

precise impartial half of the partnership… the computer. Executable computer source code is the central storage media.

For technical knowledge, the key elements of the TRAPPING process are source code, simplification, and synergism. As that source code is created, debugged, and refined, it will be mass produced at low cost on an annual revision basis. Communication among knowledge trappers, educators, and manufacturers will yield synergistic benefits. People in the region will learn what to teach, how to teach it so it stays taught, and offer the systems that make it happen… for mankind's use and enjoyment. An efficient regional industry of hardware/software mass production for A)computing and for B)human/knowledge interaction will be sustained.

CLAIMS DISCUSSION

CLAIM #1: Inter-Generational Knowledge Loss

Most people have not considered inter-generational knowledge loss to be a major problem... many have been lulled into complacency about the knowledge situation by articles in the popular press which describe how rapidly human knowledge is growing.

A key cause of the knowledge loss is that individuals choose not to reveal it. Examine your personal experience: how many masters of technology have you personally known who left their company or went to their grave without leaving anyone in their class behind? How much development work is being done in organizations to re-create capabilities and understandings that existed in the past?

Another cause of knowledge loss is disrespect for simplicity. Perhaps this isn't obvious. In my direct experience in R&D, all the things I worked on became simpler over time. Yet, the world seemed to become progressively more complex. In response to this personal paradox, I developed "Complexity Theory":

> **Q. Given that individual understanding simplifies over time, why doesn't society as a whole simplify over time? A: People are paid and promoted based on the perceived complexity of what they do. Both in government and industry, technologists usually respond by doing work that appears complex. But they squander their limited mental resources in the process and overall progress is slowed.**
>
> **Only the simple solutions to problems THAT MANY OTHER PEOPLE HAVE FAILED TO SOLVE are respected. That's because there is presently no way to judge the difficulty of solving a totally new problem. When a simple solution is found to a totally new problem, people tend to assume that it was a simple problem. The idea that it was a gifted technologist who solved it seldom enters their minds… particularly if the creative individual is not already well known. (Consider also that an "opportunity" isn't considered to be a "problem" until a lot of effort has been expended trying to realize the**

opportunity.) Thus, comparatively few technologists work on new opportunities in simple ways.

Widespread public recognition of the contributions will be a key incentive for technologists to contribute knowledge to the public domain. Previously UNTAPPED knowledge can be trapped. Being able to trap the technical knowledge of each generation would save an ENORMOUS WASTE. The mechanics of the trapping process are discussed in the next section.

CLAIM#2: Knowledge Trapping Can Be Done Efficiently.

What I propose is to TRAP and SIMPLIFY relevant applied mathematics in functional SOURCE CODE modules. Then, organize the modules in such a way that they can stand alone OR INTERACT with other modules. Note that fragments of technical knowledge are routinely trapped in software. Such software is often not clearly written and is seldom simplified.

The primary benefit of SIMPLIFICATION is that personal mental resources are freed up to consider new things. Free mental resources are the margin in which efficient progress occurs. (Complexity, which ties up mental resources, and affordable progress tend to be opposed.)

It is also important that the modules INTERACT with one another efficiently as well as stand alone. If the inter-relationships between modules are worked out properly, then an interested user can examine the connections. The user who isn't interested in the mechanics of correct interaction can ignore that aspect.

Experience has convinced me that software capability can increase much more rapidly than the number of lines of source code. Working in flight simulation, using an annual software revision/condensation approach, I found that the core source code listing stayed at about constant size while the functionality expanded synergistically. Consider the following example:

> **The last application of the simulation software (in 1984) was to study the manual docking of the Orbital Maneuvering Vehicle (OMV) with the space telescope in low Earth orbit. In retrospect, two key analytical pieces were missing: iterative orbital mechanics and double precision position. The absence of orbital mechanics was obvious... and in essence became three lines of FORTRAN. The need for double precision became obvious the first time I had hands-on experience flying the OMV... both the telescope and my vehicle were bumping along on the least significant bits defining where they were! It was visually obvious. All in all, adding about twenty lines of FORTRAN to the core software generalized the capabilities to include Earth orbit. Developing the simulation took only about two weeks.**

The approach being advocated here is different from the approach commonly used today. The prevailing approach is to develop a module and

freeze it. The (roughly) 20,000,000 bytes of disk space required by a pared-down UNIX operating system (for an HP-9000 computer in 1987) is a consequence of the 'freeze the module' approach.

It is far too early to tell how large the core source code for trapped technical knowledge might become. My long term dream is 20,000 lines of code… less than one and a half million characters (bytes) of source code… each module capable of standing alone… each model capable of interacting with other modules.

The Greater Minnesota Corporation is focusing on education because it's commonly perceived to be the core problem. But identifying what to teach deserves as much attention as improving the teaching itself. The knowledge trapping activity is concerned with condensing knowledge in a utilizable, examinable form. When that has been done well, the groundwork is laid for effective education… which is discussed in the next section.

CLAIM#3: The Mass Production of Knowledge

What I propose is the mass production of software source code and associated documentation that is so simple, so elegant, and so useful that it becomes an integral part of the technical foundation of the future… an integral part of what is known.

Mass produced applied mathematics is hard to imagine. Beyond +,-,x, & ¸, the sine and cosine of an angle are rare examples of widely accessible mathematical functions. They are so accessible (e.g. on pocket calculators) and so standardized that they are taken for granted and are essentially free. For almost all other math functions, a fairly expensive technically astute person or a fairly expensive commercial software product utilizes them.

Executable software without source code delivers capability without providing access to the underlying understanding of the process. Using this approach, many software developers hide their know-how to aid in recovery of their technical development costs. Products result, but overall there is a great deal of waste associated with knowledge hiding. Knowledge is re-created over and over… between companies, across generations, in classrooms, between nations. Competitors develop a comparable set of capability because knowing that a problem is solvable is the major hurdle to finding a solution. Those who are not in competition are denied the opportunity to apply the techniques in fields unrelated to the commercial interests of the software developers.

The technical education situation might be summarized as follows:

Human technical progress is slowed
because the analytical trails
leading to the frontiers of science
don't have readily usable maps.

I am calling into question the assumption that the U.S. will be more of an "information society" than the rest of the world by the year 2010. The sheer number of foreign technologists that we have trained (and

continue to train) in our Universities decreases the probability of that outcome. Neither knowledge trapping itself nor the associated enhanced education is likely to improve the RELATIVE vitality of the region because the educational benefits will accrue to everyone. At the same time, not pursuing them will be very disadvantageous to the region. How to serve the region's relative vitality is discussed in the next section.

CLAIM#4: Mass Production of Knowledge Interaction Equipment.

Mass production manufacturing is where the fundamental social utility of companies lies. Economies of scale support thorough product design, reduce unit manufacturing costs, improve long term product support, and encourage embedded product utilization in other systems. But mass produced SOFTWARE faces the problem that the cost of reproducing it is practically zero once the first copy is in final form. (The constructs that make it expensive, such as copyrights, are entirely artificial.) Thus, I recommend that the regional industry which mass produces the HARDWARE associated with knowledge interaction receive at least as much nurturing as the corresponding software industry. Both hardware and software mass production should be encouraged. By shaping the core knowledge here in the public sector, regional hardware and software designers and regional educators will have advance notice of new trends and can influence the trapping process itself.

It is worth considering that improving people's ability to apply knowledge can lead to more leisure time: that, in turn, can lead to a (long-overdue) four day workweek. With alternative Fridays and Mondays off, everyone would have 26 four day weekends in a year. The recreation sector of the region's economy would be one of the beneficiaries. This would be in keeping with my belief that human destiny is to have fun… because it's the one thing that computers can't do!

CONCLUSION

Knowledge is cumulative over time. By focusing on the trapping and simplification of knowledge, we can accelerate its accumulation. By timely, effective communication among knowledge trappers, educations, and manufacturers, the knowledge structures, teaching techniques, software, and enabling hardware can come together in cost effective, synergistic, mass produced systems serving a world market. Focus of TECHNOLOGY first. As the technical knowledge trapping activity bears fruit, regional interest in non-technical knowledge trapping will grow. While the region's technical advantage may be transient, the impact on cumulative technical knowledge will be permanent. The region has the option of being remembered, in history, as the place where WORLD education was born in 1990.

I sent "TRAPPING & MASS PRODUCING KNOWLEDGE" to scores of individuals; e.g., my College Physics Professor's response:

Yale University

Physics Department
P.O. Box 6666
New Haven, Connecticut 06511-8167

Campus address:
217 Prospect Street
Telephone:
203 432-3600
Facsimile:
203 432-6175
203 432-3824

April 16, 1991

Dear Mr. Litterholm,

Glad to see you're interested in education.

The next step after this might be for you to get into closer contact with the schools to find out what's really going on there and what you could do to keep the situation.

Nice to hear from you again.

sincerely

William Lechte

Handwritten because yale laid off my secretary as an economy move
W.L.L.

Appendix A: An example of trapped technical process knowledge:

Computing Discrete Fourier Transforms (DFT's).

The FORTRAN source code below is not meant to be self-explanatory. The code traps, in a very compact way, how DFT's work, and provides a software tool for performing the computation.

```
      SUBROUTINE DFT(NSAMPLES,SAMPLES,DFTCOES)
C     25 JANUARY 1993  1200L  Version  -  486:33/DrDOS/uSF5.1  -  JMS
C     Discrete Fourier Transform Implementation  (Indexed:  0:(NSAMPLES-1))

      DOUBLE COMPLEX
     . DFTCOES(0:(NSAMPLES-1))
     .,SAMPLES(0:(NSAMPLES-1))
     .,COSI(0:99)
      DOUBLE PRECISION
     . ANGLE
     .,PI2

      PI2=8.D0*DATAN(1.D0)
      IF(NSAMPLES.GT.100) STOP 'subDFT: NSAMPLES>100'
      NSM=NSAMPLES-1
C   Compute the sines and cosines used in the transformation

       DO 10 N=0,NSM
         ANGLE =PI2*N/NSAMPLES
 10      COSI(N)=DCMPLX(DCOS(ANGLE),DSIN(ANGLE))
C   Compute the fourier coefficients

    DO 30 N=0,NSM
      DFTCOES(N)=(0.D0,0.D0)
      DO 20 ND=0,NSM
        NA=MOD(N*ND,NSAMPLES)
 20     DFTCOES(N)=DFTCOES(N)+SAMPLES(ND)*COSI(NA)
 30   DFTCOES(N)=DFTCOES(N)/NSAMPLES
      RETURN
      END
```

REF: "Fourier", p. 168, Mathematics Dictionary, Ed. Glenn James & Robert C. James, New York, Van Nostrand Reinhold, 5th Ed. 1992.

Trapping how DFT's actually work may seem hopelessly esoteric to many readers. Yet, it is an elementary concept in digital signal processing. Trapping technical knowledge (through the sophomore year of college will involve trapping thousands of other processes which will appear equally esoteric to non-technologists.

"Knowledge trapping" and "educating" are somewhat distinct endeavors. Knowledge trapping concerns expressing useful, working processes in simple, compact ways. Education concerns making processes relevant, accessible, and user-friendly to students.

Establishing the trapped processes to either stand alone or co-operate is vital. The resulting integration of knowledge across technical disciplines could enhance communication between educators in different fields, and hence help unify curricula.

A FORTRAN program to exercise subroutine DFT (listed on page A-1):

```
      PROGRAM DFTTEST
C     25 JANUARY 1993  1200L  Version  -  486:33/DrDOS/uSF5.1  -  JMS
C     Program to exercise subroutine DFT

      DOUBLE COMPLEX
     . DFTCOES(0:4)
     .,SAMPLES(0:4)

      SAMPLES(0)=(.1D0,0.D0)
      SAMPLES(1)=(.2D0,0.D0)
      SAMPLES(2)=(.3D0,.1D0)
      SAMPLES(3)=(.4D0,0.D0)
      SAMPLES(4)=(.3D0,0.D0)

      WRITE(6,8) (I,SAMPLES(I),I=0,4)
    8  FORMAT(1x,'DFTTEST: Input Dataset'      ,5(/1X,I2,2F14.8))
      CALL DFT(5,SAMPLES,DFTCOES)
      WRITE(6,9) (I,DFTCOES(I),I=0,4)
    9  FORMAT(1x,'         Fourier Coefficients',5(/1X,I2,2F14.8))
      END
```

The resulting output:

```
DFTTEST: Input Dataset
  0    0.10000000    0.00000000
  1    0.20000000    0.00000000
  2    0.30000000    0.10000000
  3    0.40000000    0.00000000
  4    0.30000000    0.00000000
         Fourier Coefficients
  0    0.26000000    0.02000000
  1   -0.07411638   -0.04695718
  2    0.00138181    0.01344577
  3   -0.03666045   -0.00108509
  4   -0.05060497    0.01459650
```

<These numbers were re-generated using the FORTRAN source code and agree with the BASIC output (the original values differed).

```
10 CLS :KEY OFF :USER$="___"                                          :COLOR 15
11 PRINT "DISCRETE FOURIER TRANSFORM EVALUATION PROGRAM"              :COLOR  3
12 PRINT "Source code: DFTEP.BAS   01-25-1993 10:00 cst version  JMS":COLOR 11
13 PRINT "Runtime:"; TAB(27); DATE$; " "; TIME$; "           "; USER$:COLOR  7
20 REM ***** Define Example [%:integer, #:double precision, $:alphanumeric]
25 NDATS%=5               :REM Number of complex datapoints (5)
30 NM%=NDATS%-1           :REM       - indexing will be from 0 to (NDATS%-1)
35 TIMEINC#=.1#           :REM Time increment between datapoints (in seconds)
40 DIM SR#(NM%), SI#(NM%) :REM SR#=Signal Real part, SI#=Signal Imaginary part
45 DATA .1#,0#,.2#,0#,.3#,.1#,.4#,0#,.3#,0#  :REM the 5 points are complex
pairs
50 FOR ND%=0 TO NM%: READ SR#(ND%),SI#(ND%):NEXT ND%:REM Input data is defined
55 DIM WR#(NM%),WI#(NM%)  :REM WR#=Cosine term (real), WI#=Sine term
(imaginary)
60 DIM FR#(NM%),FI#(NM%)  :REM FR#=Fourier Real part, FI#=Fourier Imaginary
Part
65 PI2#=8#*ATN(1#)        :REM PI*2
70 REM ***** Compute the Sine and Cosine Terms Used by the Fourier Transform
 75 FOR N%=0 TO NM%
 80 WR#(N%)=COS(PI2#*N%/NDATS%)
 85 WI#(N%)=SIN(PI2#*N%/NDATS%)
 90 NEXT N%
95 REM ***** Compute the Fourier Transform
 100 FOR N%=0 TO NM%
 105 FR#(N%)=0#: FI#(N%)=0#
 110 FOR ND%=0 TO NM%
 115 NCS%=N%*ND% MOD NDATS%
 120 FR#(N%)=FR#(N%)+SR#(ND%)*WR#(NCS%)-SI#(ND%)*WI#(NCS%)
 125 FI#(N%)=FI#(N%)+SR#(ND%)*WI#(NCS%)+SI#(ND%)*WR#(NCS%)
 130 NEXT ND%
 135 FR#(N%)=FR#(N%)/NDATS%
 140 FI#(N%)=FI#(N%)/NDATS%
 145 NEXT N%
150 REM ***** Reconstruct the Signal
155 DIM YR#(NM%), YI#(NM%): REM YR#=Signal Real part,  YI#=Signal Imaginary
part
 160 FOR ND%=0 TO NM%
 165 YR#(ND%)=0#: YI#(ND%)=0#
 170 FOR N%=0 TO NM%
 175 NCS%=ND%*N% MOD NDATS%
 180 YR#(ND%)=YR#(ND%)+FR#(N%)*WR#(NCS%)+FI#(N%)*WI#(NCS%)
 185 YI#(ND%)=YI#(ND%)-FR#(N%)*WI#(NCS%)+FI#(N%)*WR#(NCS%)
 190 NEXT N%
 195 NEXT ND%
200 COLOR 14: PRINT     "Points       Sreal             Simaginary      Time D
    omain  (Input Signal)":COLOR 7
205 FOR N%=0 TO NM%:PRINT USING"####  ########.######## ########.######## ####
    ##.###### sec.";N%,SR#(N%),SI#(N%),TIMEINC#*N%: NEXT N%
210 COLOR 14: PRINT     "Coeffs       Freal             Fimaginary      Freque
    ncy Domain  (Fourier)":COLOR 7
215 HZ#=1#/(TIMEINC#*NDATS%)
220 FOR N%=0 TO NM%:PRINT USING"####  ########.######## ########.######## ####
    ##.###### Hz";N%,FR#(N%),FI#(N%),N%*HZ#: NEXT N%
225 COLOR 14: PRINT "Points       Yreal             Yimaginary      Time D
    omain  (Reconstruction)":COLOR 7
230 TIMEOUT#=1#/(HZ#*NDATS%)
235 FOR N%=0 TO NM%:PRINT USING"####  ########.######## ########.######## ####
    ##.###### sec.";N%,YR#(N%),YI#(N%),TIMEOUT#*N%:NEXT N%
240 END
```

```
DISCRETE FOURIER TRANSFORM EVALUATION PROGRAM
Source code: DFTEP.BAS    01-25-1993 10:00 cst version  JMS
Runtime:                  01-25-1993 10:28:20          ___
Points       Sreal             Simaginary      Time Domain  (Input Signal)

   0         0.10000000        0.00000000     0.000000 sec.
   1         0.20000000        0.00000000     0.100000 sec.
   2         0.30000000        0.10000000     0.200000 sec.
   3         0.40000000        0.00000000     0.300000 sec.
   4         0.30000000        0.00000000     0.400000 sec.

Coeffs       Freal             Fimaginary      Frequency Domain  (Fourier)
   0         0.26000000        0.02000000     0.000000 Hz
   1        -0.07411635       -0.04695721     2.000000 Hz
   2         0.00138183        0.01344576     4.000000 Hz
   3        -0.03666042       -0.00108510     6.000000 Hz
   4        -0.05060496        0.01459647     8.000000 Hz
Points       Yreal             Yimaginary      Time Domain  (Reconstruction)
   0         0.10000011       -0.00000008     0.000000 sec.
   1         0.19999994       -0.00000000     0.100000 sec.
   2         0.29999998        0.10000001     0.200000 sec.
   3         0.39999997        0.00000002     0.300000 sec.
   4         0.29999993       -0.00000001     0.400000 sec.
```

An Exercise: To appreciate the utility of the source code, try computing the Discrete Fourier Transform for the signal data (in the box above) without using this appendix as a resource. Use public library resources as a starting point. Wait to compare your DFT coefficients with the ones above until after you're sure you've got the right answer.

There are several hurdles to cross:
1. Finding the Discrete Fourier Transform equation.
2. Writing an Algorithm that implements the equation.
3. Programming the equation in a compatible computer language
   That supports the math operations of the algorithm.
4. Debugging the resulting software. (Note, there is some variation in the scaling of DFT coefficients; the values shown above were normalized by the number of samples… see lines 135 and 140 in the basic program.)

The above steps involve dealing with information in several different disciplines. Most people don't have the time to work out these details to the point of being confident of having the correct solution. Manually computing the very simple five point signal given here is not routinely practical. Computers will be used to compute DFT's; *as a tool, DFT's are not taught until they are automatically computable*.

The essence of the transform, as a tool, is expressed in nine lines of FORTRAN on page A-1. The compact representation is well suited for embedding the transform in other pieces of software. Educators encountering DFT's for the first time should be comforted to know that the implementation can be so simple.

LINKS:

My subsequent *technical* knowledge trapping activities include:
"quantitative Visual Presence":
http://ftp.setterholm.com/Geodesy/quantitativeVisualPresence11.pdf & (2009)
http://ftp.setterholm.com/Geodesy/qVPMath12-AppendixA-20091211.pdf
"Homogeneous Optics":
http://ftp.setterholm.com/Optics/H-Optics08.pdf (2010)
"Hyperspace Algebra Tools":
http://ftp.setterholm.com/PseudoInverse/Hat.pdf (2011)
~A 3D(Stereo) Homogeneous Visualization Environment:
http://ftp.setterholm.com/3DEnvC/3DEnv.pdf (2016)

An overview of my website:
http://ftp.setterholm.com/Directory.pdf (2016)

A meta-philosophy (version 2.0) encompassing the ideas proposed herein:
http://ftp.setterholm.com/Philosophy/LTDW20.pdf (2004)
Robust philosophy is the foundation of robust decision making.