On Folk Theorems, and Folk Myths

Peter J. Denning

There is an extraordinary article, "On Folk Theorems," by David Harel, in the July 1980 issue of the *Communications*. It makes the amazing observation that there exist in the conventional wisdom deep theorems which are simple, intuitive, widely believed, and of obscure origin. These propositions are undissolved lumps in the porridge of knowledge we regard as the folklore of our field.

To the delight of the layman and the dismay of the formalist, a folk theorem is usually oversimplified. It is so credible that most of us would believe we could prove it within a short time—even though in actuality the proof may be difficult. Most of us would call the result "well-known." Few of us know who was the first to discover it.

Harel illustrates these points with a study of the familiar

FLOWCHART THEOREM. *Every flowchart is equivalent to a* **while***-program with one occurrence of* **while-do***, provided additional variables are allowed.*

Now, I make it a point to be familiar with the origins of important ideas in computer science. So, my initial reaction on reading this theorem was: "How did this get past the reviewers? I saw this exact statement in Knuth's paper in *Computing Surveys* in December 1974." But the next several pages showed me how little I really knew. Harel had sleuthed painstakingly through no less than ninety-one papers and books dating back to 1936! He left little doubt that the flowchart theorem has many roots; it is the distillation of many results. Harel's article may well serve as a definitive reference on a result of murky origin.

Are there other folk theorems? (Or, is the Flowchart Theorem the only example?) Contrariwise, are there Folk Myths—*false* propositions which are simple, intuitive, and widely believed? I believe so.

Four More Folk Theorems

My first nomination for another folk theorem comes from programming distributed computations; it relates a simple syntactic condition to the overall determinacy of a set of parallel processes sharing data:

DETERMINATE PARALLEL PROCESSES THEOREM. *The final values of shared variables are uniquely determined by the initial values provided that variables modifiable by any given process are disjoint from variables accessible to any concurrently executable process.*

My second nomination is familiar to programmers of many sorts:

SORTING THEOREM. *Optimal sorting algorithms take time proportional to* n *log* n *to sort* n *items in the worst case.*

This statement is strictly true only for sorting algorithms that compare one pair of elements at a time. If parallel comparisons are possible, the worst-case sorting time can be made proportional to n.

My third nomination has been long of interest to designers of page

replacement policies for virtual memory systems:

OPTIMAL PAGING THEOREM (for fixed size memory allocation). *The page causing a page fault should displace the resident page that will not be used again for the longest time.*

Practical policies approximate this idea by estimating, from past usage, the time until next reference of each page resident in the main store.

My fourth nomination has been of no little interest to performance analysts:

LITTLE'S FORMULA. *The mean length of a queue is the product of the throughput and the mean response time.*

This result was used informally long before J.D.C. Little showed in 1961 that it is formally correct. Many alternate proofs have been published since then.

In each of these cases, the theorem statement is intuitive and easily believed; it oversimplifies but is essentially correct. The simplicity of each statement belies the difficulty of a formal proof. There is a vast literature leading up to and refining each result.

Folk Myths

Folklore has traditionally been hospitable equally to truths and myths. Even so in computerlore: there are *false* propositions which are simple, intuitive, and widely believed. These myths can be remarkably resilient, holding sway even in the face of hard scientific evidence to the contrary.

Here is an example: Suppose we observe that a set of tasks can be executed on a set of parallel processors in a certain amount of time. Most people believe that if we increase the number of processors, or reduce the execution times of some of the tasks, or remove ordering constraints among some of the tasks, then the set of tasks can be completed sooner. In fact, it is possible for any of these "improvements" to *increase* the time to complete all the tasks!

Here is another: Most people believe that increasing the amount of main memory allocated to a program will reduce the paging rate in a virtual memory system. In fact, it is possible for some paging algorithms to *increase* the amount of paging when more memory is allocated to some programs. (Curiously, such paging algorithms are common in real operating systems.)

Here is still another: Many people believe that worst-case behavior of algorithms is a good indicator of average behavior. As an example, consider algorithms for the fast retrieval of record indices when the entire index table can be stored in main memory. It is tempting to put n record indices in a tree so that the worst-case lookup time is proportional to $\log n$; it is tempting to avoid storing record indices in a hash table because the worst lookup time can be proportional to n. In fact, the average lookup time in the tree is also proportional to $\log n$, whereas in the hash table the average is a small constant independent of n.

Here is an example more subtle and insidious: Many people believe that it is easier to overcome poor software performance by waiting for the technology to make the same hardware faster and cheaper than to redesign the software or hardware. Thus, poor performance of virtual memory systems is dealt with, not by engineering more efficient memory management policies, but by adding more memory or by hoping (futilely) for the day when the entire storage system will be monolithic. Unreliable software is dealt with, not by hardware architectures with simple run time error checking, but by increasingly complex compilers that require the faster hardware to run at the same speed. This myth's ability to impede progress and block innovation is proven. (Ah, an excellent subject for a future article.)

In each of these cases, the actual behavior runs counter to the expected (intuitive) behavior—and yet the actual behavior is often characterized as anomalous!

* * * *

It is fascinating that our young field has so many fine examples of important ideas with long, complex histories. Few of us appreciate the impact of the past, the intricate ways different paths of thought have intertwined to produce the present. We can learn much by heeding history. We can be much more skeptical, lest beguiling myths continue to detour us from the path of progress.