On Building Systems That Will Fail

Pooja Makula[1], *SridharChimalakonda*[2]
Indian Institute of Information Technology, Chittoor, Sri City, A.P., India
pooja.m13@iiits.in[1],*sridhar$_c$h@research.iiit.ac.in*[2]

***Abstract*—In this article, Fernando J. Corbato describes how the complexity of large innovative and ambitious projects will always lead to mistakes. He explains this by drawing a few examples. Ambitious systems often tend to fail because there is always a change and a new challenging problem. He also talks about the various properties of the ambitious systems and how developers should assume that any given error will occur at any given point of time and therefore plan how to handle it.**

***Index Terms*—**

I. Introduction

A. Ambitious Systems - What are they?

Ambitious systems are that class of systems that never quite work as expected. Things usually go wrong sometimes in the most unexpected and dramatic ways. These systems that fail are really much more common than we think.

Sometimes we feel that the failures of these ambitious systems are due to the human act and that they can be corrected by at least building the technical parts of the system correctly. When it comes to the computer systems in particular, it is only a matter of getting the code debugged, and some people assume that by doing rigorous testing the job will be done. But unfortunately there are many cases where none of these techniques like proving program correctness or testing will always work in making the systems efficient.

Here is an examples of ambitious systems that will usually fail:

- Boston driving - The Boston area is notorious for the free interpretations which drivers make of these pesky regulations and perhaps the epitome of it occurs in the arena of the traffic rotary.They offer great opportunities for creative improvisation, thereby adding zest to the sport of driving. The only reason there are not more accidents is that most drivers have a second component to the strategy, namely, they assume everyone else may be crazy they often are right and every driver is really prepared to stop with inches to spare. Again we see an example of a system where ambitious tactics and prudent caution lead to an effective solution.

B. Properties of Ambitious Systems

Some of the general properties of ambitious systems.
{i} First - They are usually vast.
{ii} Second- They are complicated most of the time and are too much for even a small group to develop.
{iii} Third - They take a lot of time to build.
{iv} Fourth - They can sometimes become so important that they cannot be thrown away.
{v} Fifth - They are never finished!They invite a flood of improvements and changes.

II. Ambitious Systems - Technological Changes

We have seen galloping growth over a period of four decades and it still does not seem to be slowing down. The field is not mature yet and already it accounts for a significant percentage of the Gross National Product both directly and indirectly. More importantly the computer revolution, this second industrial revolution, has changed our life styles and allowed the growth of countless new application areas.

A. What is the Problem?

According to the author, these seem to be the few problems,

- The main reason why ambitious systems can get right is change. The computer field is intoxicated with change.
- This change and growth has not only changed the world we live in but has raised our expectations, spurring on increasingly ambitious systems in such diverse areas as airline reservations, banking, credit cards, and air traffic control etc.
- Computers are tailored for special applications and for parallelism. Parallelism is not a solution for every problem. Certain calculations that are intrinsically serial, such as rocket trajectories, have very limited benefit from parallel computers.

B. A few Technological Changes

As we approach the present, corresponding to a personal computer, the graph really should become more complicated since one consequence of computers becoming super-cheap is that increasingly, they are being embedded in other equipment.

- Previously the "machine room" was staffed with one lone operator. The boxes are huge, shower stall sized, and the overall impression is of some small factory.
- Another reminder of the immense technological change which has occurred is in the physical dimensions of the main memories of computers. For example, if one looks at old photographs taken in the mid 1950's of core memory systems, one typically sees a core memory plane roughly the size of a tennis racquet head which

could hold about 1000 bits of information. Contrast that with today's 4 megabit memory chips which are smaller than one's thumb.

Then the author talks about his work on the two pioneering time-sharing systems, CTSS and Multics. He says that these two systems can be taken as the examples of ambitious systems and he also gives the reasons why the complexity of the tasks involved made it almost impossible to build the systems correctly the first time.

III. CTSS - The Compatible Time-Sharing System

The problem with the systems that existed in the 1960's was that Users, i.e. programmers, were expected to submit a computing job as a deck of punched cards; these were then combined into a batch with other jobs onto a magnetic tape and the tape was processed by the computer operating system. The problem was that even for a trivial input typing mistake, the job would be aborted. Time-sharing, was the solution to the problem of not being able to interact with computers. So they developed their own version of the time-sharing vision and they called it The Compatible Time-Sharing System. The main aspirations were:

- First - It is meant to be a demonstration prototype.
- Second - General purpose programming was possible.
- Third - It was possible to run most of the large body of software that had been developed over the years in the batch-processing environment.

A. *CTSS: Problems Faced*

The basic scheme used to run CTSS was that, the supervisor program, which was always in main memory, would commutate among the user programs, running each in turn for a brief interval with the help of an interval timer. But there were definitely a few difficulties

- The key difficulty was that main memory was in short supply and not all the programs of the active users could remain in memory at once. Thus the supervisor program not only had to move programs to and from the disk storage unit, but it also had to act as an intermediary for all I/O initiated by user programs. Thus all the I/O lines should only point to the supervisor program.
- The supervisor program had to prevent user programs from trampling over one another. To do this required special hardware modifications to the processor such that there were memory bound registers that could only be set by the supervisor.
- There were no standard terminals, simple modems, character based i/o. no lower case, interrupt timers nor standard calendar clocks.
- There was no CPU memory protection nor boundary registers.
- There was no way to prevent user programs from issuing raw I/O instructions at random nor there was an easy way to store large amounts of data with relatively rapid random access.

B. *CTSS: Results*

- Interactive use was a different mode for users.
- General - purpose was important for upgrades. - develop new supervisor software using the system itself
- Compatible software meant a running start.
- Users had persistent storage.
- Remote operation became the norm.
- Users began to share more programs and data more. - keeping information on-line in the central file system suddenly made it especially convenient for users to share and exchange information among themselves.
- New terms of user frustrations developed. - users who had been enduring several hour waits between jobs run under batch processing, were suddenly restless when response times were more than a second.
- System limitations began to increase. - Suddenly the issues of privacy, protection and backup of information had to be faced.

C. *CTSS: Observations*

{i} First - It became quite influential and lasted more than expected.
{ii} Second - The then-new transistors and large random-access disk files were absolutely critical to the success of time-sharing. The previous generation of vacuum tubes was simply too unreliable for sustained real-time operation and, of course, large disk files were crucial for the central storage of user programs and data.

D. *A Mishap*

When we have a multitude of novel issues to contend with while building a system, mistakes are inevitable. And indeed, the same was faced with CTSS. The problem was when CTSS was being used as the main time-sharing workhorse, any user who logged in, found that instead of the usual message-of-the-day typing out on his terminal, he had the entire file of user passwords instead.
To simplify the organization of the initial CTSS system, a design decision had been made to have each user at a terminal associated with his own directory of files. Moreover the system itself was organized as a kind of quasi-user with its own directory that included a large number of supporting applications and files including the message-of-the day and the password file. Normally a single system programmer could login to the system directory and make any necessary changes. But the number of system programmers had grown to about a dozen in number, and, further, the system by then was being operated almost continuously so that the need to

do live maintenance of the system files became essential. Not surprisingly, the system programmers saw the one-user-to-a-directory restriction as a big bottleneck for themselves.

IV. Multics

The Multics system was meant to do time-sharing "right" and replace the previous ad hoc systems such as CTSS.

A. Multics: Innovations

- CPU Addressing:
  {i} Paging and segmentation with access control.
  {ii} Ring scheme of memory protection.
  {iii} n processors, m memory units.
  {iv} system programmed newly defined compiler language PL/I.

- Radical Hardware changes meant:
  {i} no assembler, compilers, software and conventions of modularity.

B. Multics: Observations

- Novel Hardware: software had to be built from ground up.
- PL/I a new language: hence a new compiler technology needed.
- Three organizations only coupled by cooperation: no easy way to reject weak ideas.
- System strong points: virtual memory, file systems, security, on line reconfiguration, file backup system.
- System was ambitious: misled by the ease of developers - earlier systems which were built "brick-by brick, predictions about projects never done before are really only guesses.
- System was late
- Unix was a reaction to Multics: Bottom up implementation vs top down design, still no close.
- Honeywell developed a commercial base of about a hundred sites worldwide, many of which are still in use.

V. Sources of Complexity

The general problem with ambitious systems is complexity. A few major causes mentioned are:

- Source:
  In particular, the larger the personnel required, the more levels of management there will be. The difficulty is that with more layers of management, the top most layers become out of touch with the relevant bottom issues and the likelihood of random serendipitous communication decreases.
  Large projects encourage specialization so that few team members understand all of the project. Misunderstandings and miscommunication begin, and soon a significant part of the project resources are spent fighting internal confusion. And, of course, mistakes happen.
- New Design Domains:
  The most vivid examples come from the world of physical systems, but software too is subject to the same problems albeit often in more subtle ways.
- Human Usage:
  {i} Privacy and security of data and procedures. {ii} Backup, Archiving and Data Bases. {iii} The usual discussion of risk and uncertainty is flawed.
- Rapid Change:
  The change which is often driven by technology improvements. A result is that changes in procedures or usage occur and new vulnerabilities can arise. For example, in the area of telephone networks, the economies and efficiencies of fiber optic cables compared to copper wire are rapidly causing major upgrades and replacements in the national telephone plant. Because one fiber cable can carry at a reasonable cost the equivalent traffic of thousands of copper wires, fiber is quickly replacing copper. As a result a transformation is likely to occur where network links become sparser over a given area and multiply interconnected nodes become less connected.
- Multiplicity of technologies:
  {i} Evolution of technology: telephone lighting, autos, radio, movies.
  {ii} Lifestyle changes: TV editing, view graphs, ATM's, word processing, answering machines, FAX, e-mail
  {iii} Consequences: More filters and trust are needed to cope, changes lead to confusion and mistakes.

VI. Conclusion

- Simplicity and Elegance:
  Elegance is the achievement of a given functionality with a minimum of mechanism and a maximum of clarity.
- Metaphors:
  Metaphors have the virtue that they have an expected behavior that is understood by all. In effect metaphors are a way of internalizing and abstracting concepts such that one's thinking can be on a higher plane and low-level mistakes are avoided.
- Constrained Languages:
  Use of constrained languages for design or synthesis is a powerful methodology. By not allowing a programmer or designer to express irrelevant ideas, the domain of possible errors becomes far more limited.
- Anticipate Errors of Use:
  One must try to anticipate both errors of human usage and of hardware failure and properly develop the necessary contingency paths. This process of playing "what if" is not as easy as it may sound since implicit is the need to attach likelihoods of occurrence to events and to address issues of the independence of failures.
- Design for repair and evolution:
  It should be assumed in the design of a system, that it

will have to be repaired or modified. The overall effect will be a much more robust system, where there is a high degree of functional modularity and structure, and repairs can be made easily.

- Cross-educate development team:
  One of the best investments that can be made is the cross-education of the team so that nearly everyone knows more than he or she needs to know. It really is a case of "more heads are better than one."

VII. Final Words

First, the evolution of technology supports a rich future for ambitious visions and dreams that will inevitably involve complex systems.

Second, one must always try to learn from past mistakes, but at the same time be alert to the possibility that new circumstances require new solutions.

And third, one must remember that ambitious systems demand a defensive philosophy of design and implementation. Or in other words, "Don't wonder if some mishap may happen, but rather ask what one will do about it when it does occur."

References

[1] Corbató, F. J., Daggett, M. M., and Daley, R. C., "An Experimental Time-Sharing System," Proceedings of the Spring Joint Computer Conference, M

[2] Corbató, F. J., and Vyssotsky, V. A., "Introduction and Overview of the Multics System," Proceedings FJCC, 1965.

[3] David, E. E., Jr. and Fano, R. M. "Some Thoughts About the Social Implications of Accessible Computing," Proceedings FJCC, 1965.

[4] Corbató, F. J., Clingen, C. T., and Saltzer, J. H.,"Multics: The First Seven Years," Proceedings of the SJCC, May 1972, pp. 571-583.

[5] Brooks, F. P., Jr. "No Silver Bullet," IEEE Computer, April 1987, 10-19