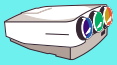
# INFRASTRUCTURE RISK REDUCTION

Harold ”Bud” Lawson
Lawson Konsult AB
Lidingö, SWEDEN



bud@lawson.se

**What will be addressed?**

- The Problem of Platform Stability
- Historical Perspective

(Or - It did not have to happen this way)

- What to do about the Situation?

## BACKGROUND

- Today's computer and communication systems contain significant unnecessary complexity
- These complexities permeate platforms of hardware and system software components
- The platforms host many complex, often critical, applications

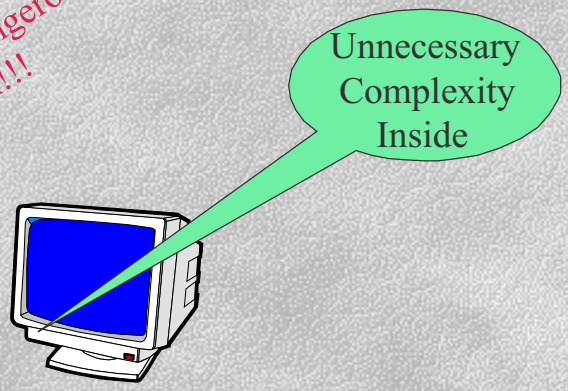
## Two main sources of Unnecessary Complexity

§ Feature-itis - (Too much functionality)

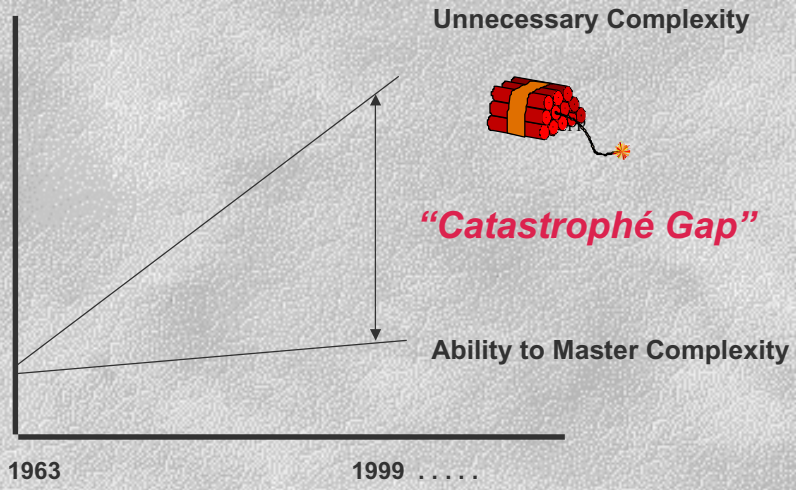
§ Levelwise mapping of application functions - via programming languages, operating systems, protocols and middlewares onto poor or inappropriate platforms

## TRUTH IN ADVERTISEMENT

Smoking may be dangerous to your health!!!




## REAL DANGER of the CURRENT SITUATION



## A quote from Intel Fellow Robert Colwell

- *"Users will require better dependability and security.*

  *Antilock brakes that "mostly work" or "hardly every crash" wouldn't be acceptable, but that describes general-purpose computing today.*

  *This situation arises because we do not design hardware in conjunction with software, application developers don't design software with the OS, and companies place less emphasis on the overall hardware-software system reliability than in getting to market quickly.*

- *Ultimately the lack of system dependability could well become industry's concern because it will become society's burden."*

## ”REMEMBER THE YORKTOWN”

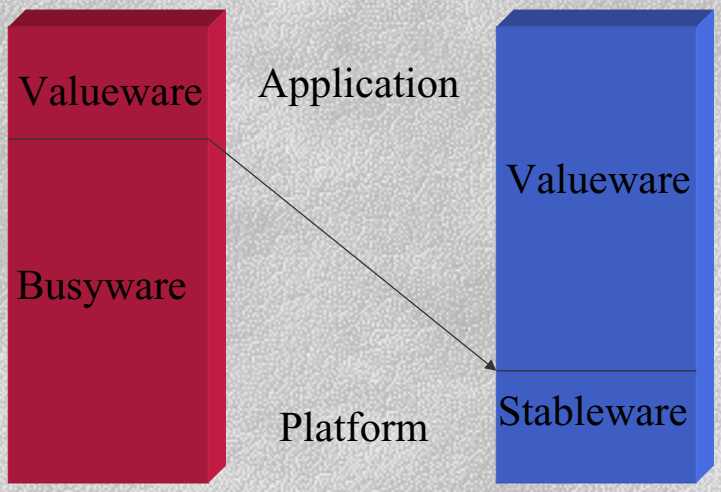
A Windows NT error caused the disabling of the Aegis guided missile cruiser Yorktown.

The Yorktown could not perform its mission and had to be towed into port.

”This is a prime example of an otherwise well engineered system that failed due to a platform bug.”

Most likely, there are many platform related failures in a variety of critical application domains that are not publicly reported.

## THE ESSENTIAL TRANSFORMATION




## *Busyware Characteristics*

- **Complex mapping of application functions - via programming languages, operating systems, protocols, middlewares - onto poor or inappropriate platforms**
- **Perpetuates the motivation to use low levels of programming (including C, C++ and Java)**
- **Results in unnecessary "costs" and "risks" (especially in maintenance)**
- **Focus on "mundane knowledge" of isolated aspects (resulting in enormous remuneration for trivialities)**

## *Stableware Characteristics*

- **Promotes true high level programming by closing the language-machine semantic gap**
- **System software functions provided via well defined "machine" semantics**
- **Language and system machine semantics are standardized, verifiable, and certifiable**
- **Stimulates competition in providing Stableware**
- **Enables concentration on and establishes a strong market for "good" Valueware**

## *Conclusion*

**Mainstream computing has proven to be useful**

**-HOWEVER-**

**It has lead to Busyware and thus is dangerous**

## Any Salvation?

**Alternative architectures (have been) can be developed that (could have been) can be applied in realizing verifiable and certifiable Stableware**

**Achieving the transform is technically feasible**

When will the transform occur?

Most likely, it probably will only happen when one or more major catastrophe's occur

For example
not being able to maintain 40 million + lines of code, a major incurable virus, and/or crashes in major systems (finance, transport, …).

Far-sighted companies could plan for it now and get a jump on the market

## Historical Perspective

*(Or - It did not have to happen this way)*

Hardware mainstream computing devices

Dominated firstly, by the IBM 360/370 family

and

secondly, by the Intel X86 family

## The Weakening of Architecture

IBM 360/370 and Intel x86

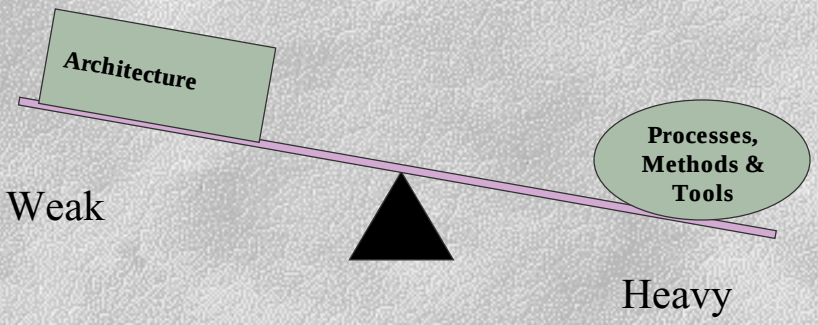
Both ingenious from a circuit technology viewpoint

Instruction sets source of significant complexity

Made system and application software complex

Widened the Semantic Gap

## *Tipping the Balance*



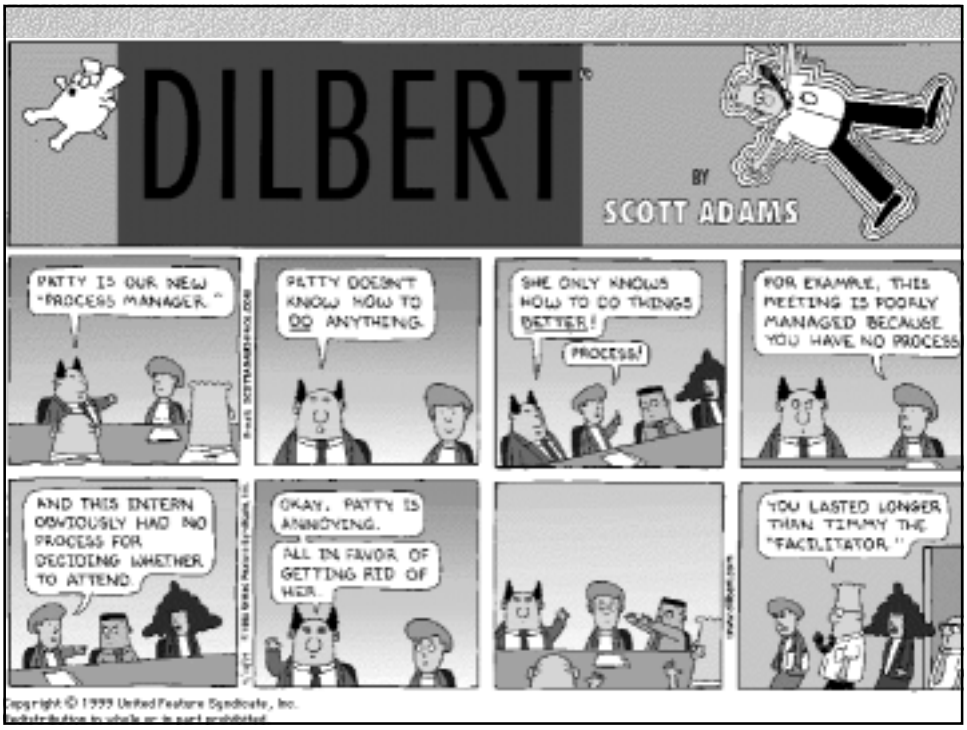

An Emphasis of Heavy Processes and their Evaluation

Appeals to people who do not know what they are doing.

Creates information overload: thus compounding the problem of unnecessary complexity.

Mastery of even small portions of the complexity creates a deep feeling of satisfaction without solving the real problem.

"see Dilbert Cartoon on the Process Manager"




**Programming languages**

Fundamental tools of the programming trade.
Fred Brooks points out:

*"The transition from machine level to higher level languages led to the single biggest productivity gain ever made in software development."*

*Therefore: It is surprising and alarming that today's system programs, composed of megabytes of code, are constructed using low levels of programming abstraction including C, C++ and Java.*

## Operating systems

In the IBM 360/370 era, OS/360 was born.

"Eighth Wonder of the World".
A case a galloping complexity.

Complexities arose that were beyond the comprehension of the mortals involved.

Fred Brooks, manager of this effort, developed notion of the "Mythical Man Month."

After inheriting OS/360 management; Watts Humphrey developed his view of the need for "Capability Maturity."
An approach to ”Mess Management”.

## Systems Engineers

a new lucrative profession was born

With the altruistic goal of helping the customers solve their application problems with the new computer technology, IBM introduced Systems Engineers.

Reality

Systems engineers job existed because customers did not have a chance of mastering System/360 complexity.

## Attempts to Reduce Complexities

Late 1950s and early 1960s, work of
Robert Barton leads to Burroughs B5000 and B5500.

Stack oriented machines provide
a language friendly instruction set.

Compilers for Algol and COBOL
straight-forward to implement.

Due to the hardware technology at that time,
machine was difficult to realize and expensive.

## General-Purpose Microprogrammable Processors

In the late 1960s and early 1970s

Standard Computer Corporation MLP-900,
Nanodata QM-1, Burroughs B1700, Datasaab FCPU

Provide microprogramming environment for
multiple instruction set emulation as well as
language friendly instruction sets.

Not long after the microprocessor entered the marketplace and
by the late 1970s, the X86 era had begun.

Again, an end to creative architectures aimed at reducing
the hardware - programming language semantic gap.

## What About RISCs and CISCs?

Late 1970s and early 1980s arguments:
RISCs and CISCs

Several advantages with RISC approach

Bottom line, X86 (a CISC architecture) became
the mainstream hardware device

While better, RISCs not particularly
programming language friendly

Complex code generation decisions are
passed up into compilers

## MULTICS Operating System

Implemented in the PL/I programming language

MULTICS, developed at MIT, had commercial life with
General Electric and Honeywell

Experience with MULTICS based systems
indicated manageability of installations

In contrast, UNIX is implemented in C and
we observe concrete results of complexities in managing
and operating incompatible variants of UNIX installations

## Improving upon the situation

Lift the level of programming abstraction for all systems and application programming

Require the definition of programming languages and key system functions as standardized, verifiable ”machines”

Close the semantic gap between programming languages and system functions in respect to the instruction repertoires of microprocessors

Effectively employ levelwise function distribution as a means of reducing total system complexity

## A new generation of microprocessors

Introduce the general purpose microprogrammable processor as the mainstream processor work horse

RISCs and CISCs firstly joined by, and eventually replaced by FIMs - (Function Integration Microprocessors)

Based upon general purpose microprogram technology with multiple emulation facilities

Sound basis for achieving the necessary improvements
-programming language friendly instruction repertoires
-functions migrated from higher levels into microcode

Attempts were made to move this way earlier
Why should they succeed this time?

Firstly, combined hardware, system software complexity limit for current X86 technology is rapidly being reached

Secondly, due to advances in circuit technology, there is no limiting hardware resource factor

Thirdly, CAD tools are available to permit us to generate the hardware designs at a fraction of the cost and time

Finally, FIM design is significantly simpler than the complex pipelined structures of the X86 era

## CONCLUSIONS

- Unnecessary Complexity makes mainstream computing risky business
- Improvement is technically feasible
- Suppliers of platforms must take “limited” product responsibility
- Standards for programming languages and platform critical functions must be well defined “as machines” that are verifiable and that can lead to certification of suppliers product

## The Transform from Busyware to Stableware can (and must) be made!!!

A "mini-step" towards deploying FIMs
was taken in the pico-Java machines

An XMLstandard where parsers and interpreters
are clearly defined as machines could provide a
natural starting point in moving towards Stableware

*Any Venture Capitalists Listening!!!*