Robert Brotzman 7084

NSA Seminar Time / Format 1/14/69

Time	Topic	Who	interval
9^{00}	What is Multics; Why is there a management problem	Saltzer	30 m
9^{30}	Sensitive issues in management of large systems	Corbató	45 m
10^{15}	Discussion	"	30 m
10^{45}	Coffee	—	15 m
11^{00}	Controlling a Software Research Project: Multics experience	Saltzer	45 m.
11^{45}	Discussion	"	30 m
12^{15}	Lunch	—	1 hr.
1^{15}	PL/I as a tool	Corbató	45 m.
2^{00}	Discussion	"	30 m.
2^{30}	Coffee	—	15 m.
2^{45}	Management of Operation of Multics-type Systems	Saltzer	30 m.
3^{15}	Discussion	"	30 m.
3^{45}	Wrapup; Problems of the future	Corbató	15 m.
4^{00}	Seminar formally ends; informal discussion		

Could interchange

Introduction N.S.A. Seminar 1/15/69

Seminar Title: "Management of Development of the Multics System"

also: Implications on Operation

Format: Corbató, Saltzer alternating
lectures followed by 30 minute Discussion periods.

Short Breaks for coffee at $10^{\underline{45}}$
and $2^{\underline{30}}$

Break for lunch $12^{\underline{15}}$ → $1^{\underline{15}}$

End at $4^{\underline{00}}$ promptly

We begin with short introduction to What Multics is and why there is a management problem.

Follow immediately with "Sensitive Issues in Management of Large Systems"
(1st of four lectures)

What is Multics?   Series of images

Multiplexed Information and Computing Service.

A research project to explore the implications of the computer utility – by building one.

Joint Project – MIT Project MAC ; Bell Labs, GE

support of ARPA

Going into 5th year – will take on significant users [illegible] 3Q69, Sys Prog [illegible] 1-2Q69

Scale of project : 4 years, ~ 30 man so far.

Successor to Compatible Time Sharing System (CTSS) on IBM 7094.

What does it do?   images

"Large scale time-sharing system" init implentation GE 645

~~Long-term file~~ information storage system. for long-term reliable retention of user files.

Ability to interact with program.

What is new?   Multics provides several features not available in other systems

1. Programmer sees a virtual memory

directly addressable - like core

which encompasses not only his program and his immediate data

but also all information stored in the "file system"

Put another way, the ~~semantics of file accessing are merged with the semantics of array accessing.~~ class of information structures known as a "file" ~~are~~ is no longer a special class; ~~but is treated by the programmer~~ "file" is identical with "array" and is accessed as such.

Special hardware is needed to make this concept effective.

Concept is of particular interest in problems using a large data base, ~~which [illegible] fit in core~~

2. Information sharing / Privacy

Allow (where wanted) sharing in core memory of "segments" of information.

Working through the full implications of controlled sharing introduces considerable complexity.

(Don't need to explain why privacy is wanted.)

Sharing allows

1. Many users to simultaneously work on a single data structure.
2. Users to trivially borrow each others utility programs.
3. Two users executing same subroutine need only have one copy in core. (Supervisor falls out as a particular case of a set of shared procedures dealing with a shared data base.)

3. Modularity / Expandability / Reliability in hardware through pools of identical major modules.

CPU's

Core boxes

Disks / Drums , etc.

Includes "Multiprogramming with Multiple Processors" to maximize equipment utilization.

4. Ability to guide the course of a computation from an interactive console. One can dynamically (during execution) guide a computation to any procedure in the system; planning in advance, though possible, is not required.

e.g., can run a procedure with a missing subroutine as long as it isn't called. If it is called, can designate what to do. May save computation while you write the subroutine.

5. Removing the distinction between console and batch jobs.

interactive   absentee

Same command (or job control) language

Same supervisor

Same I/O conventions

Same character set.

Ability to switch back and forth with impunity.

There are many other smaller features in Multics

Main distinguishing feature: ~~it does all these things~~ ~~at the~~ that a single system combines all these features.

Thus: interactions produce inevitable complexity.

Virtual memory

+

Virtual access method

+

Sharing / Privacy

+

Modular hardware

+

Dynamic computation guidance

+

many small features

all at once ⇒ Complexity

⇓

Research

not dealing with proven ideas.

1. New Hardware architecture (paging / segmentation / I-O / clock / Drum)
2. New Operating system
3. New Implementation language.

Links to firm specs:

PL/I subset only

635 slave mode instruction set

Leaves a very large variable space in which to search for solutions.

<u>Why</u> is there an unusual management problem?

1. This is a <u>research</u> project, not merely development based on well-known and proven ideas. We are trying to do something for the first time.

~~ex~~ a. New Hardware b. New System c New Language

2. System is too complex to permit managers who do not understand technical aspects <u>better</u> than the people working directly on the programs.

3. Highly technical people are too scarce and unreplaceable to use punishment for not meeting schedules

4. If a project takes a long time (e.g., 4 years) there must be special incentives to insure management continuity – It is unusual for a dynamic, capable manager to stay in one place for more than two years.

Let us now launch into the real meat of todays discussion with a discussion of "Sensitive Issues in Design of Large Systems."

1/14/69 NSA Lecture #1 (Saltzer)

Controlling a Software Research ~~Prg~~ Project: Multics Experience.

→ Basic assumption: Initial design will be reworked.

Three phases: Control ~~strategies~~ tactics evolve as system point in order.

Initial design

Coding / unit check

System shakeout, functional / performance / reliability.

Initial Design

a. Small design team, which has responsibility for guiding later implementation. (Incentive to make sure ideas are workable) Small size issue discussed earlier.

b. Documentation as a ~~pr~~ preoccupation

allows later transfer of ideas to others ← this point is usually emphasized

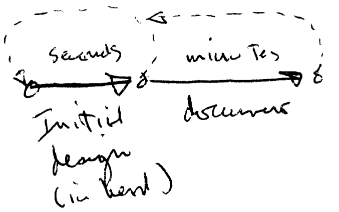
permits a check that everyone agrees on what is being done. } these are more important

frequently flushes out fuzzy thinking.

c. Design review - the documentation must be read as well as written.

(Build up design : feedback about need for redesign.)

turn-around time to get going after a flaw is discovered.



Coding / Unit check phase

a. ~~[illegible]~~ Progress reporting: Segment Inventories, etc.

(Problem: visible evidence of progress is nil:

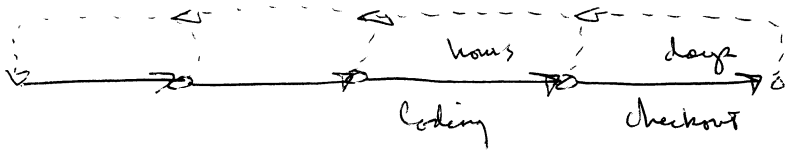
~~4~~. no documentation pouring out

no system running.)

b. (Discovered need) Design review after coding, but before large investment of resources to checkout and integrate.

Presumption: i Many implications of spec only appear ~~after~~ during code phase, but may be unnoticed by programmer.

ii An overview at this stage can detect unnoticed global problems, and call attention to decisions not anticipated by other users of a module.



<u>System shakedown phase</u>: Is the system coming out right?

a. Need to separate Functional progress
Performance "
Reliability "

See new b.) on next page

~~b. Need to Certify successive versions of system.~~
~~i. to be sure a new feature or module has not adversely affected performance.~~
~~ii. to be sure changes have not introduced obvious bugs.~~

c. Need special tools to explore performance (remember: initial design is presumed non-optimum - now we begin to optimize)

i. Segment Usage metering

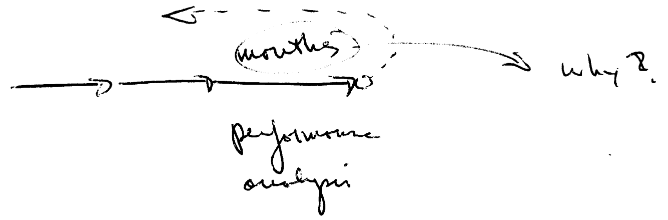
ii. Supervisor subroutine execution time measurement.

iii Repeatable load (PDP-8 or internally driven scripts)

iv. periodic reports to user about resources used (Ready message)

v. Probe + display, to watch system.

vi. Script representing model user...



0. Not easy to discern real source of a performance problem.

1. Flawed routine deeply embedded, others use it.

2. Programmer is cold on project.
~~[illegible] Problem may be subtle [illegible]~~ all encompassing

6. Need to control rate of change

With many programmers, not all of equal quality, they will introduce new bugs as they work.

Danger: with a reasonably large staff every new system may [illegible] consist entirely of changed procedures.

Administrative throttles are necessary, to funnel all changes past a certification team, which tries them in small batches rejects changes which either

reduce performance

cause system to fail certification tests.

Comments on Reworking plan

Primary assumption: a *research* project means you are doing something *new*. In software it means new functions, more functions, new techniques.

(New Hardware, New System, New Language)

⇓ implication

First version of any design is probably not optimum.

View: Reworking of modules is part of the plan. It is *not* a sign of disaster.

Biggest trick is to discover *need* to rework as early as possible: but you can't catch em all.

(refer to earlier diagram.)

CPU time required to handle a "missing page fault"

1. allocate core space, throwing out a page
2. figure out where the missing page is
3. start I.O and give CPU away (Multiprogram)

$$\frac{N}{2} \leftarrow N \leftarrow 3N \leftarrow 30N \leftarrow 60N$$

more unified strategy (still in EPL)

simplified functional view of problem.

initial poking around to locate difficulty, some EPL tuning

Estimated could get to N/2 by going to modern language.

Amount of code required to implement interprocess communication library routine

$\overset{12}{\cancel{12}}N_{words} \longrightarrow \overset{3}{\cancel{10}}N_{words} \longrightarrow 3N_{words}$

functional redesign

better understanding of how it plugs into system + improved use of EPL.

There are perhaps 10 such examples of key modules.

Only sensible view is:

The module has not really been completely specified until the first coded version is complete:

Why? Because there is no other methodical way of assessing all of the implications of the

external specs.

planned internal structure

programmer's tactics

and all of their interactions.

Implication: 1. Design review after coding

2. Don't give up if it needs a rewrite - expect it.

Observation: this view is parallel to situation in other research areas.

Most striking analogy → 1. Algebra / Math theorem proving.

2. Hardware design goes through a breadboard phase.

3. Course notes go through many drafts.

4. Fortran compiler is now a relatively easy job - it didn't used to be

It is not considered incompetence to not choose the best path first.

(Management / Psychological Implication)