Robert Bratzman

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^{45}
and 2^{30}

Break for lunch $12^{15} \rightarrow 1^{15}$

End at 4^{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 <u>Multics</u>?

series of images

Multiplexed Information and Computing Service.

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

Joint Project – <u>MIT Project MAC</u>; Bell Labs, GE

support of ARPA

Going into 5th year – Will take on significant user load 3Q69, Sys Prog load 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" 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, ~~where~~

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 some ~~one~~ subroutine need only have one copy in core. (Separation 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 some computation while you write the subroutine.

5. Removing unnecessary distinctions between console and batch jobs.
   - console → interactive
   - batch → absentee

Some command (or job control) language
Some supervisor
Some I/O conversion
Some character set.
Ability to switch back and forth with ease.

There are many other smaller features in Multics

Main distinguishing feature: ~~it does all these things at once~~ has a single system combining all these features.

Thus: interactions produce inevitable complexity.

Why is there an unusual management problem?

1. This is a research project, not merely development based on well-known and proven ideas.
   We are trying to do something for the first time.
   a. New Hardware b. New System c. New Language
2. System is too complex to permit managers who do not understand Technical aspects better than the people working directly on the programs.

n.b. Technical constraints were more important than cost and time

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 today's discussion with a discussion of "Sensitive Issues in Design of Large Systems."

Switch to Cobol

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

→ Basic assumption: Initial design will be reworked.

Three phases: Control ~~of~~ design tactics evolve as system passes 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 ~~pre~~ preoccupation

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

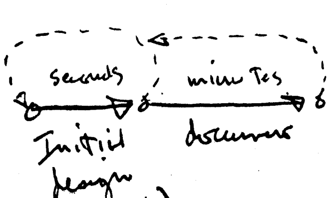
permits a check that everyone agrees on what is being done. } these are more important

frequently flushes out fuzzy thinking.

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

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

turn-around time to get going after a flaw is



Coding / Unit check phase

a. ~~Management~~ Progress reporting: Segment Inventories, etc.

(Problem: visible evidence of progress is nil:

~~b~~. 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 implementation flaws only appear

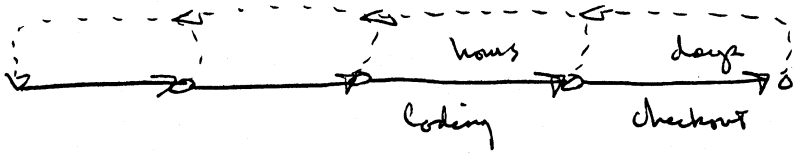
during ~~after~~ code phase, but may be unnoticed by programmer.

ii An overview at this stage can detect

unnoticed global problems, and call

attention to decisions not ~~understood~~

but other users of a module.



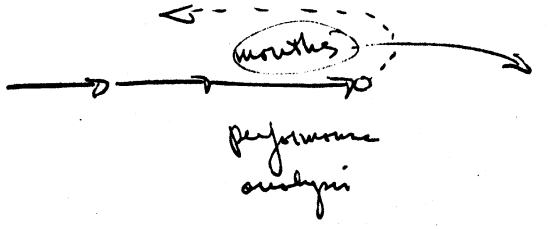
System shakedown phase: Is the system coming out right?

a. Need to separate Functional program
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 premature optimization — now we begin to optimize)
i. Segment Usage metering
ii. Supervisor subroutine execution time measurement.
iii Repeatable load (POP-8 or internally driven scripts)
iv. periodic reports to users about resources used (Weekly 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 called on project.

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 ~~have~~ contain certain of changed procedures.

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

reduce performance

cause system to fail certification test.

Comments on Rewriting plans

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

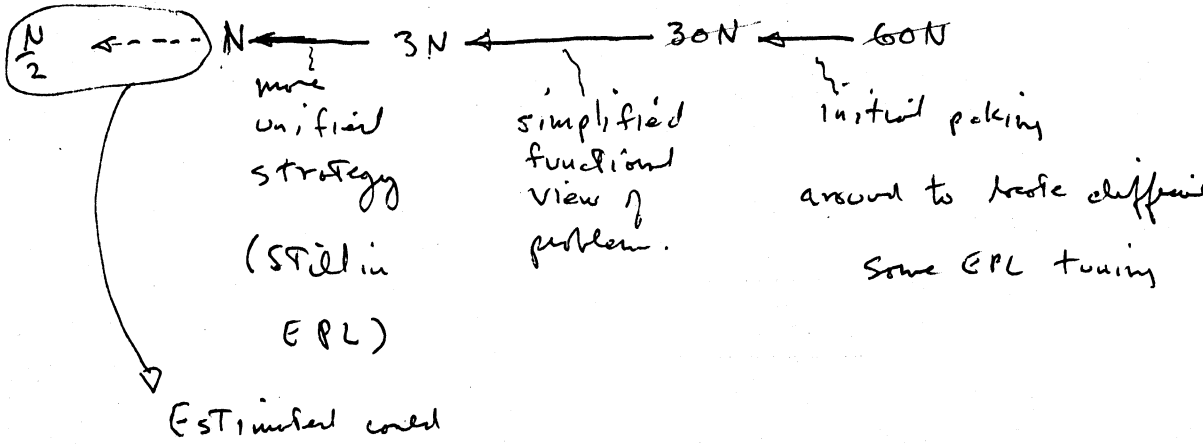
First version of any design is probably not optimum.

view: Rewriting of modules is part of the plan. It is not a sign of design error.

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

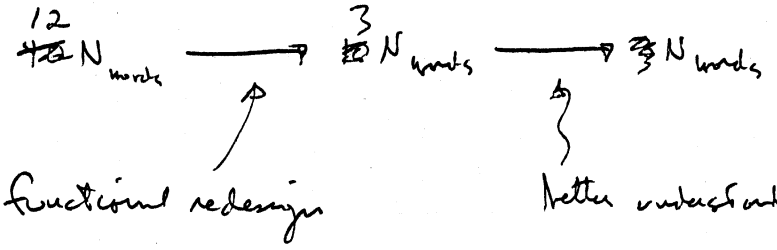
(refer to earlier discussion.)

· 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)



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

Amount of code required to implement interprocess communication library routine



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.

Implications: 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. Computer notes go through many drafts.
4. Fortran compiler is now amazingly good — it didn't used to be

It is not considered incompetence to not choose the best path first.
(Management / Psychological Implications)

Implications on Operation

The Impact of sophistication on other aspects of the operating system and its environment is great. For example.

1. There tends to be a bigger gulf between a system and its administrators. The guy who has to decide to order $ more core is frequently ~~unable to~~ not in a position to know why it is needed. He needs simple guidelines for complex decisions.

2. The operating staff must be turned up.
   i. Potential for a disaster is high (eg. - a single missed switch could screw up file system and require reload of all information)
   ii. As users become anonymous people-at-a-distance, operators find they are alone in the machine room - at almost always.
   iii. Operators become guardians of privacy - must mount proper tapes; assess etc.
   iv. operations soon takes on aspects of a Telephone exchange.

3. A first-class maintenance team is required.
But what is their incentive?
Must give them challenging jobs near heart of system to keep their interest and willingness to spend much time going after dull bugs.

4. Tension of changing systems goes up.
   i. Sheer problem of viewing a new system - large type, etc.
   ii. Many more opportunities for a small change bug to have hidden effects.
   iii. Utility dependence discourages frequent risk of exposing users to new systems.

Let us look at the various operating implications

a. A security force, which challenges the system's claim that it is protecting privacy.

b. A complaints bureau and a responsive consultative staff, with contacts through the system programmers.

c. A locksmith, who can untangle mistakes in use of privacy achieving mechanisms (e.g., lost password)

d. Administrative policy / planning / ordering / accounting functions.

e. Editorial boards, to decide which user interfaces should go in the library. (Applied with on-line, shared information systems.)

f. Off-line communication paths - newsletters, etc., to keep people informed of each others work so they can share easily.

g. Liaison with telephone communications people - (You have to learn how to live with the telephone system - it isn't going to adapt to you.)

h. Auditors, who verify that internal usage accounting is fair and accurate.

i. System measurement and tuning to keep it matched to the load presented by the user community.

Observation: Operating Momentum is very high — you can't drastically change a system like this overnight, e.g., continuity of service is important — you can't be down for six weeks while changing computers.