SOFT 437

Software Performance Analysis

Ch 7&8:Software Measurement and Instrumentation

Why do we need data?

- Data is required to calculate:
  - Software execution model
  - System execution model

We assumed that we have required data to calculate these models

SPE Data Requirements

- Key performance scenarios
- Performance objectives
- Execution Environments
- Software resource requirements
- Computer resources requirements

Type of software resources

- CPU usage
- SQL (`select <fields> from table where key=x`)
- File I/O
- Messages
- Logging to file or data bases
- Calls to middleware functions
- Calls to software in a different process/thread/processor
- Delay for remote processing
- Workload intensity (# of requests, # of users, etc)

Types of computer resource requirements

- CPU WorkUnits (`KInst or time {sec, msec}`)
- Number of physical I/Os
- Number and size of network Msgs
  - (M_i=d/(m-h)), d=data, m=size, h=header
- Delay (for remote processing)
- Service time

Authenticate Transaction Scenario

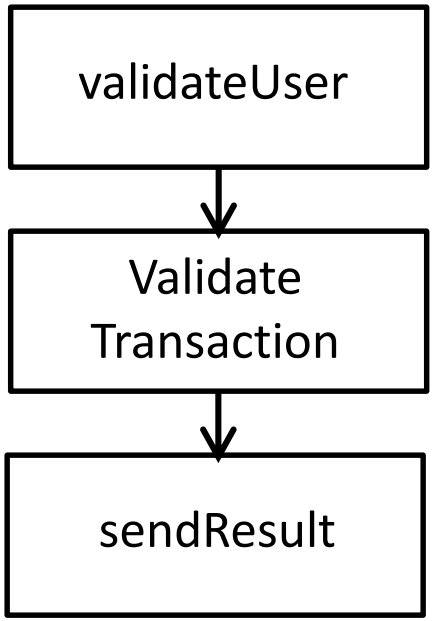
Hardware Resources

Mapping between software resource requirements and computer device usage

Table 2: Processing Overhead

Device	CPU	Disk	Network
Quantity	1	1	1
Service Unit	Kinstr.	Ph.I/O	Msgs.
Proc. U	20	0	0
DB	500	2	1
Msgs	10	2	1
Service time	0.00001	0.02	0.01

Processing Overhead Matrix



Processing Steps

Proc. U	1
DB	2
Msgs	0

Proc. U	2
DB	3
Msgs	0

Proc. U	2
DB	1
Msgs	1

Software Resource Requirement Matrix

Performance Measurement Concepts

Measurements

- Provide input data for SPE models, verify and validate models
- Determine whether performance goals have actually been met
- Monitor the performance of system over their life time

It is much easier to make measurements than to know exactly what you need to measure

Static vs. Dynamic Measurements

- Static measurements are made without executing the software
  - For example, measure the size of database table or a network message
- Dynamic measurements require that the software be executing during the measurement
- The measurements needed for SPE are primarily dynamic rather than static

There are tools that can predict the execution time of code without running it, but these are modeling tools- not measurement tools

Terminology

- A ***state*** reflects the information about the systems hardware or software within a period of time when the computer system is running
  - For example, "***idle***" or "***executing job X***" are states of the CPU
- An ***event*** corresponds to a change in the system's state
  - For example, the CPU changes from "idle" to "executing job X"
- There are two fundamental measurement techniques corresponding to states and events
  - ***Monitoring*** states
  - ***Recording*** events

Monitoring States

- A performance ***monitor observes*** the activities of a computer system and ***records*** performance information about states of interest
- Monitors ***sample*** the ***system's state*** by **periodically activating and recording the current state** along with the appropriate performance data about that state
- The data collected by the monitor allows you to identify "***hot spots***" – those portions of the code that consume most of the CPU time

Types of Monitors

- Software monitors: programs that executes independently from the software being measured
- Hardware monitors: external devices that are attached to the computer system hardware via external wires or probes
- System monitors observes the state of the overall system
- Program monitors observes the states of the particular program being measured

Recording Events

- Record the occurrence of particular events along with the appropriate performance data
- For example, recording the number of accesses to the ATM account databases, keep track of
  - The access every time one occurs
  - The requested process, the time of the request, the amount of data returned, the amount of CPU time required, the number of physical I/Os, the elapsed time for the completion of access

Types of Event Recording

Even recording can occur at different levels:

- ***Program event recording***: the events are pertinent to the program being measured
  - the CPU time for database access, the elapsed time to send a message
- ***System event recording***: certain events are recorded for all executing programs of interest.
  - the CPU time for database accesses for teller and bank analyst transactions,

Internal vs. External

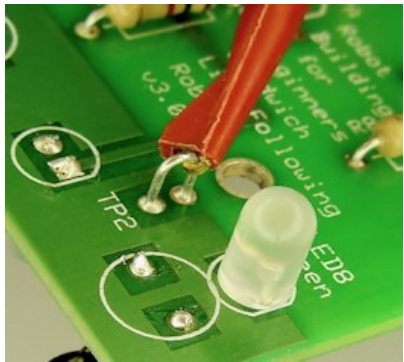
- Both monitors and event recorders may be either integrated into a program (***internal***) or ***external*** to it
- ***Internal measurement*** requires that code be inserted into programs to detect events and record the pertinent performance data
  - Such as, compiler option or a preprocessor
- ***External monitors*** and event recorders execute independently of the software to be measured

Granularity

- A range of granularity for measurements can be
  - the initiation and completion of processes, individual programs, class methods, or individual statements within methods
- There are three choices for managing the amount of data collected
  - Record all details
  - Summarize the data as it is collected and record the results periodically
  - Tally data between events and record it when an event occurs

Instrumentation

- Performance data provided by the standard tools is not always what you really need.
- Instrument systems as you build them to enable measurement and analysis of workload scenarios, resource requirements, and performance objective compliance
- Instrument software by inserting code (probes) at key points to measure pertinent execution characteristics
  - At the start and end of a business task



Reasons for Instrumentation

- There are at least three reasons for supplementing the standard tools with instrumentation:
  - Conveniently gather exactly the data you need, especially for deducing the locality of data reference
  - Match the data granularity as you need at different level (e.g., simple nodes, expanded nodes in software execution graphs)
  - Control the measurement process by turning selected instrumentation code on and off

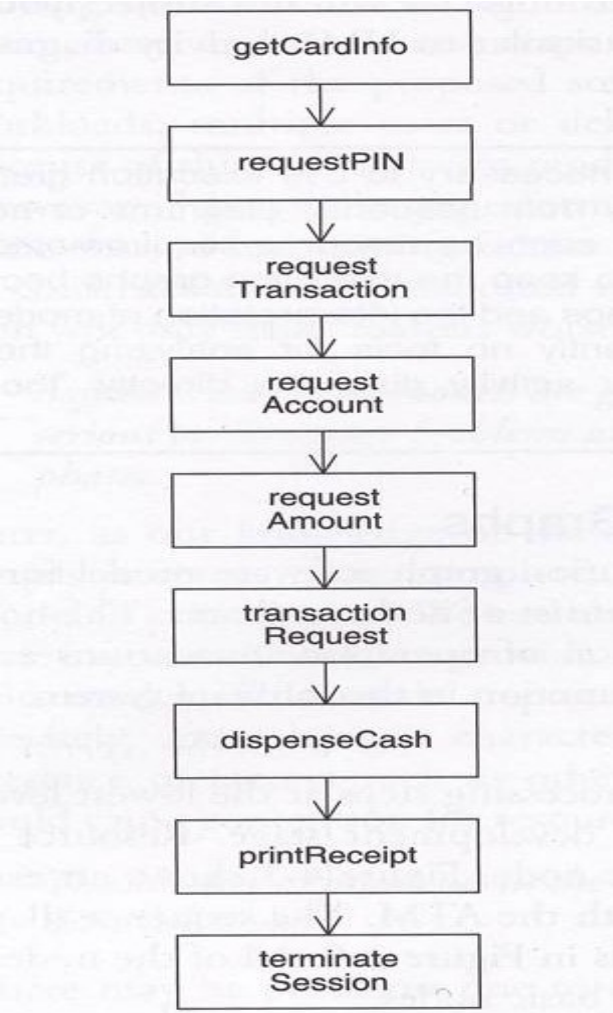


Figure 4-1: Sample Execution Graph

Instrumentation Design Considerations

1. Defining the events to be measured
   - The events are important for each scenario, including the beginning and end of key functions (critical processing steps)
2. Choosing the granularity of the measurements
   - The finest granularity is to record all events and metrics (such as process id, time stamp, etc), but at the greatest cost
   - A coarser granularity is to selectively record the data at varying levels of detail (such as major components, major and intermediate-level components, or all components)

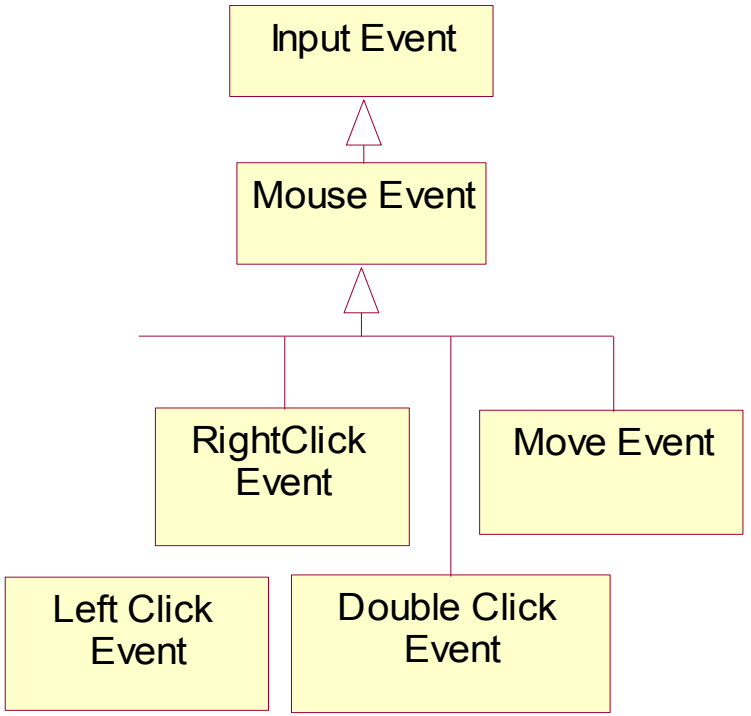
Instrumentation Design Considerations (con't)

- The coarsest of the event-recording techniques is to define types or classes of events, tally the metrics for each, and then compute the averages, variances or distributions of the metrics

3. Dynamically selecting the data to be recorded
   - Record data at runtime
   - Use instrumentation parameters to vary the metrics collected and their granularity

Parameterizing the Instrumentation

- Define hierarchies of events and /or data
- Use the instrumentation parameters to trigger the recording of classes of events or data in the hierarchy



Implementation

```
#DEFINE WRITESPEC "%2d%2d%2d%3d$%s\n"
EventLog::record(char eventName[32]){
    SysTime time = SysTime::GetCurrentTime();
    fprintf(theFile, WRITESPEC, time.getHour(),
    time.getMinute(), time.getSecond(),
    time.getMSecond(), eventName);
}
If (instrumentationON)
    theEventLog->record(eventName)
```

Steps in Conducting Measurements

1. Understand the purpose of the measurement (questions & hypothesis)
2. Identify the data needed to answer those questions, data collection tools and measurement techniques
3. Identify experimental variables and controls
4. Define the test cases: workload, software, and the environment for each test
5. Execute the tests and collect the data
6. Analyze, interpret, and present the results

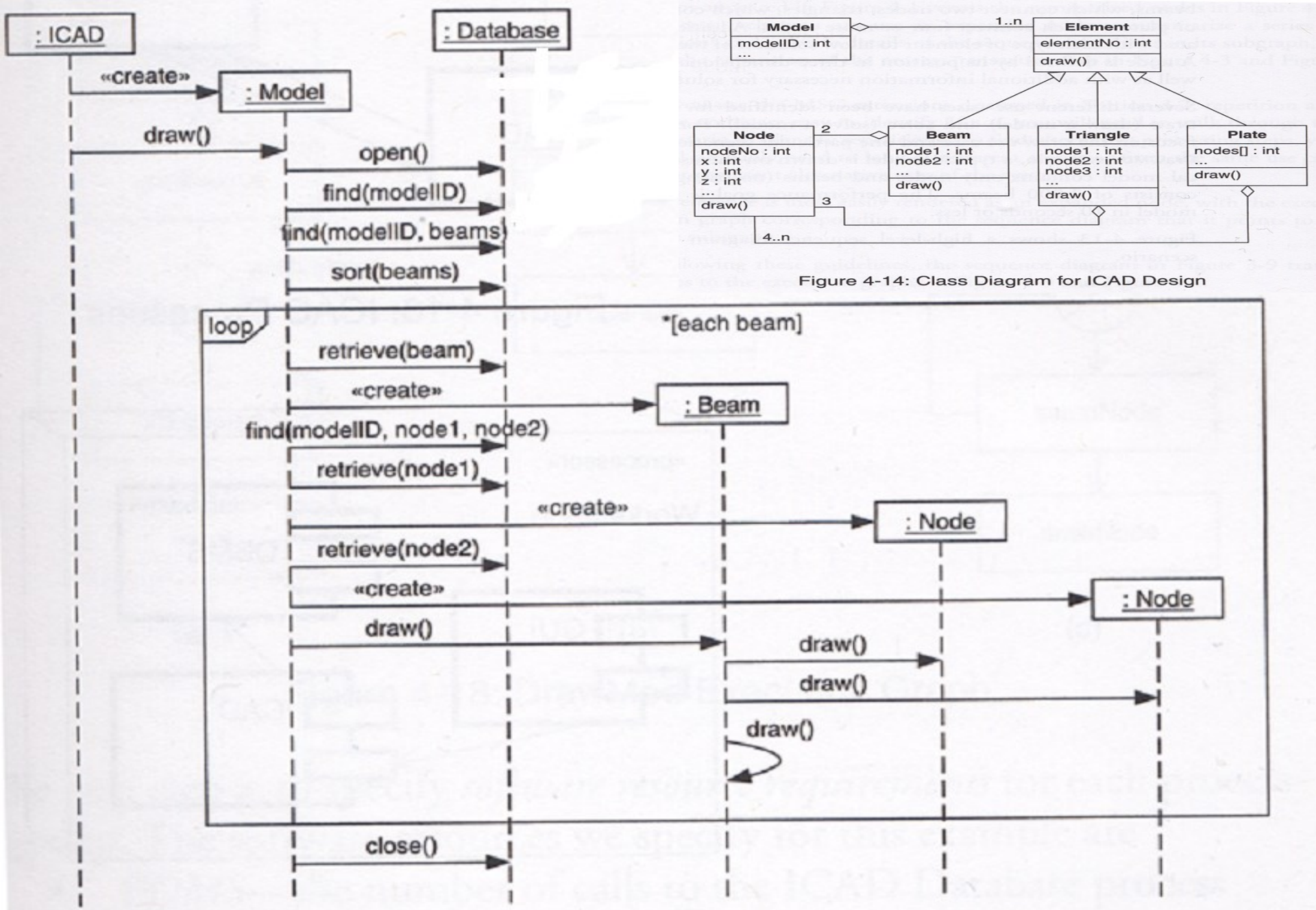


Figure 4-14: Class Diagram for ICAD Design

Figure 4-15: Expanded DrawMod Scenario

Example of Performance Measurement Design

- Define Purpose: use a standard sort algorithm vs. build a customized one
- Data Collection
  - measure resource requirements (i.e., CPU time, I/O, memory) for each sorting algorithm
  - Use a system event recorder
  - Evaluate the overhead for running a system event recorder

- Experimental variables:
  - The number of items
  - The size of the items
  - The original data order
  - Sort algorithm itself

- Test cases:
  - Isolation vs. normal processing
  - List length: 36, almost sorted
  - List length: 2,000, almost sorted
  - List length: 8,400, almost sorted
- Executing the test: run each test case multiple times

Software Measurement and
Instrumentation

Purpose of Software Measurements

- System understanding: measurements of ***existing systems*** provide performance information about ***a similar system***, or ***an earlier version*** of the system being planned
- Model specification: measurements of ***experiments, prototypes, or similar system*** provide ***workload data*** and ***estimates of resource usage***
- Model updates: measurements of ***evolving software*** replace ***earlier specification estimates*** to improve model precision

Purpose of Software Measurements (con't)

- Model verification and validation
  - Compare the SPE models to actual performance
  - Demonstrate performance-objective compliance or specific problems requiring improvement
- Software performance evaluation
  - Monitor software performance (e.g., SLA)
  - Identify areas for improvements upon deployment

Types of Measurement Data

- Workload data: the ***number*** of requests for each workload function, the ***rate*** of requests, ***patterns*** of requests
- Data characteristics: the ***amount*** of data and the ***size*** of each data item, ***frequency*** of requests for each data item, the ***locality*** of data references
- Execution characteristics include:
  - Path characteristics: the number of times ***each significant path*** is executed to compute the ***loop repetitions***, and ***the execution probabilities*** for ***conditional paths***

Types of Measurement Data (con't)

- Software resource usage: ***the number of time*** each software resource requested, ***the average elapsed execution time*** (e.g., network messages, database SELECTS, and so on)
- Processing overhead: ***the amount of service*** the software resources require from the ***key computer system resources (i.e. CPU, I/O, Network, etc.)***

- Computer system usage includes
  - Scenario response time: ***the end-to-end elapsed time***
  - Scenario throughput: the number of times that the scenario is executed per unit of time

Types of Measurement Data (con't)

- Key computer system resource usage: the actual service time + wait time
- Resource utilization: the percentage of time busy
- Resource throughput: the rate at which the resource completes service requests
- Resource queue length: the average number of jobs waiting for service

Performance Measurements	SPE Uses				
	System Understanding	Model Specs	Update Models	Model Verification and Validation	Evaluate Software Performance
Workload Data	yes	some	yes	yes	some
Data Characteristics	yes	some	yes	yes	some
Path Characteristics		yes	yes	yes	some
Software Resources	yes	yes	yes	yes	some
Processing Overhead	yes	yes	yes	yes	yes
Computer System Usage				yes	yes

Performance Measurement Planning

- Results from performance measurements must be
  - ***Representative*** and ***reproducible***
- To be ***representative***, measurement results must accurately reflect each of the factors that affect performance
  - Workload, software, and computer system environment
- To be ***reproducible***, the workload, software, and computer system environment must be controlled, in order to repeat the measurement and get the same (or very similar) results each time

Performance Benchmarks

- Performance benchmarks are standardized performance measurements
- Performance benchmarks produce data for
  - Hardware or software system evaluations
  - Volume testing
  - Performance measurement
- Benchmarks may be established for comparative purposes within an industry, or you can design benchmarks for your own software

Summary of Performance Measurements

- Load-driver measurements
  - Simulate user behavior under loading conditions
  - Run in isolation, and time consuming and labor intensive
- Performance models are preferred for most SPE tasks
- Performance measurements are
  - Not cost effective for design trade-off studies
  - But useful for deriving performance specifications, verifying and validating performance models, and confirming SPE conclusions

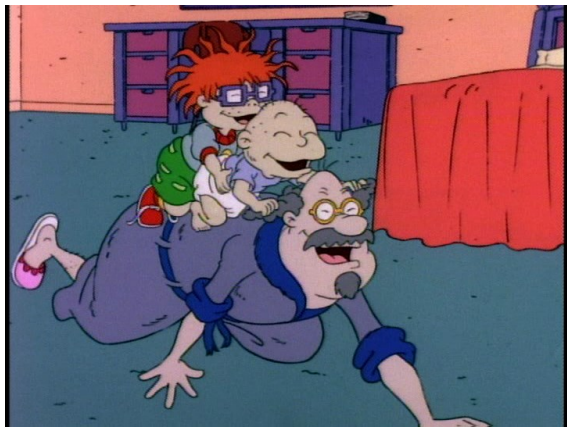
Factors Affecting Measurements

- System Perturbation
- Capture Ratios
- System Overhead
- Measurement Timing
- Reproducible Results
- Representative Time Periods
- Averages for Typical Behavior
- Workload Generation

System Perturbation

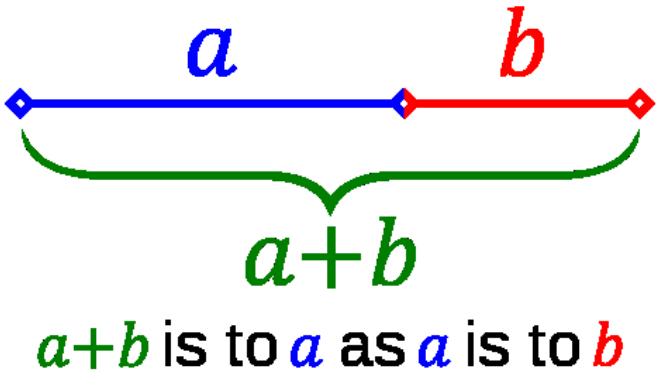
make matters worse

- The measurement process may perturb the system being measured, thereby affecting the results
  - Happen often with system measurements
- Perturbation is most significant when the system already has severe performance bottlenecks
- Control the system perturbation by creating
  - selective granularity of measurements
  - selective activation and deactivation



Capture Ratios

- The capture ratio characterizes the percent of time that the measurement technique account for either directly or by inference
- In the period of monitoring, a higher-priority process may be executing, which delays the monitor execution slightly
- Monitors should determine that there was such a delay, the length of delay, and cause for the delay



System Overhead

- A monitor may try to apportion the execution time of operating system routines
  - CPU scheduling overhead, page faults
- Include both program processing and system overhead in the software execution models

Measurement Timing

- The duration of the events and states must match the resolution of the ***operating system clock*** used to time them (HW clock vs OS clock)
- The ***sampling interval*** for monitors must be short enough to detect the states of interest, but not too short (HD clock =1GHz (1nanos), OS clock = 10ms)
- When the duration of events and states is much shorter than clock ticks, ***measure many events*** (perhaps in a loop) to obtain more meaningful results

Reproducible Results

- Other work that executes on the computer system influences the performance metrics
- Exclude the other work during measurement periods, particularly for reproducible results
- Control the measurement environment, the workload, the time of day, duration of the measurements to collect reproducible results

Representative Time Periods

- Many of the metrics provided by measurement tools are averages over the measurement period
- However, averages may not be meaningful if the period is too long
  - For example, peak loads may be 50% or higher than average loads
- Collect measurements for representative time periods

Averages for Typical Behavior

- Performance of many components varies, depending on
  - the state of the system
  - the processing required
  - e.g., Database SELECT
- To measure the time required for an individual behavior, run measurements ***long enough*** to get ***an average***, ***minimum***, and ***maximum*** that are representative of the behavior

Workload Generation

- Determine how you will produce the workload that you will measure
- If the performance of the system is dependent on user intersections, use a load driver (a script) to mimic user interactions
  - Vary the simulated arrival rate of request
  - Replay the script to get reproducible results

Example of Synthetic Workload Generation Program

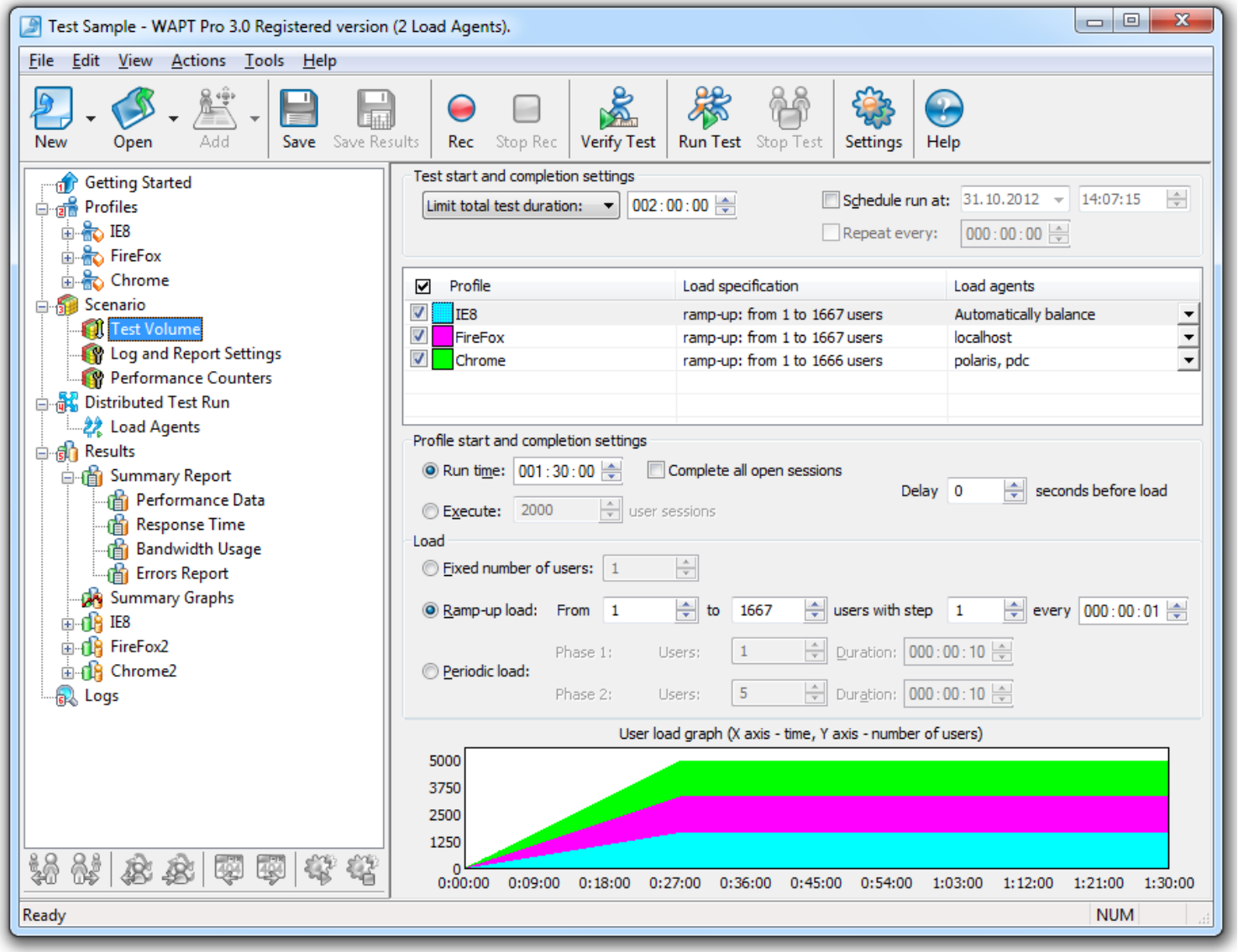
Buckholz, 1969

```
      DIMENSION Record(500)
!Control parameters:
      Num_Computes=500        !Repeat count for computation
      Num_Reads=35            !Number of records read
      Num_Writes=40           !Number of records written
      Num_Iterations=1000     !Repeat count for the experiment
!Open files:
      OPEN(UNIT=1,NAME='In.dat',TYPE='Old',
     1FORM='Unformatted',ACCESS='Direct')
      OPEN(UNIT=2,NAME='Out.dat',TYPE='New',
     1FORM='unformatted',ACCESS='Direct')
      CALL Get_Time(CPU1,Elapsed1)    !Record starting time

      DO 500 Iteration=1,Num_Iterations

!Perform a number of read I/Os
      DO 100 i=1,Num_Reads
      READ(1'i),Record
100   CONTINUE
!Do computation
      DO 200 j=1,Num_Computes
      DO 200 i=1,500
200   Record(i)=1 + i + i*i + i*i*i
!Perform a number of write I/Os
      DO 300 i=1,Num_Writes
      WRITE(2'i),Record
300   CONTINUE
500   CONTINUE
      CALL Get_Time(CPU2,Elapsed2)    !Get ending time
!Close files:
      CLOSE(UNIT=1)
      CLOSE(UNIT=2)
      CPU_Time=(CPU2-CPU1)/Num_Iterations
      Elapsed_Time=(Elapsed2-Elapsed1)/Num_Iterations
      TYPE *,'CPU time per iteration is ',CPU_Time
      TYPE *,'Elasped time per iteration is ',Elapsed_Time
      STOP
      END
```

WAPT - Web Application Load, Stress and Performance



Guppy-PE

Guppy-PE , a programming environment providing object and heap memory sizing, profiling and analysis

A fish swimming in Python

cProfile: deterministic profiling for CPU

```
$ python -m cProfile -o profile_output search.py
```

```
$ time python search.py
# real 0m1.521s
# user 0m1.250s
# sys 0m0.142s
```

Data Collection Techniques

- System monitors
  - observe ***system-level states***
  - require detailed information on ***operating system data structures and algorithms*** to identify the states and record the pertinent data
- Program monitors
  - Are less dependent on the operating system
  - May be programming-language dependent
- System event recorders
  - Provided as an operating system service

Data Collection Techniques (con't)

- External program event recorders
  - Trace program execution by recording sequences of events
- Internal event recorders
  - Insert code into the software collect performance data about events of interest

Performance Metrics	Data Collection Technique				
	System monitor	Program monitor	System event recorder	External program event recorder	Internal event recorder
Workload Data function:					
Function requests			X		X
Rate and pattern			X		X
Data Characteristics:					
Type and number					X
Size					X
Locality					X
Path Characteristics:					
Execution probability		X		X	X
Loop repetitions		X		X	X
Software Resources:					
Type and requirement	X	X			
Elapsed time		X			X

Performance Metrics	Data Collection Technique				
	System monitor	Program monitor	System event recorder	External program event recorder	Internal event recorder
Processing Overhead:					
Scenario usage	X	X	X		X
Component usage		X			X
Resource requirements	X	X	X	X	X
Computer resource usage:					
Response time	X	X	X	X	X
Throughput	X	X	X	X	X
Resource service time and wait times	X	X	X	X	X
Resource utilization and throughput	X	X	X	X	X
Queue lengths	X	X	X	X	X

Implementation Alternatives

- Integrate instrumentation specifications into the program and data specification
- Use special classes to collect and record data
  - Design a class which provides routines for data collection
  - To collect data, programs will call methods belonging to this class
  - Example in next slide
- Use standard system event recording tools

Designed-in Instrumentation

- Design instrumentation into the software
  - Define performance requirements and thus software instrumentation requirements as part of the software architecture
  - Collecting and analyzing data tends to incur less processing overhead if the probes are integrated as the design evolves
- The designed-in instrumentation
  - Eases the measurement after the system is implemented
  - Collects data for testing, diagnosing the cause of problems, and quantifying the execution cost of each requirement