Managing System Performance

System performance directly affects users. Centralized operations are easier to measure than complex networks and client/server systems. Various statistics can be used to assess system performance. Capacity planning uses operational data to forecast system capability and future needs

- Performance and workload measurement
- Response time
- Turnaround time
- Throughput

Response time

Response time is the overall time between a request for system activity and the delivery of the response. Response time includes three elements

1. The time necessary to transmit or deliver the request to the system.
2. The time the system needs to process the results.
3. The time it takes to transmit or deliver the results back to the user

Response time is critical for user satisfaction

Turnaround time

Turnaround time measures the efficiency of centralized computer operations, which still are used for certain tasks, such as credit card processing. Turnaround time is the amount of time between the arrival of a request at a computer centre and the availability of the output for delivery or transmission

Throughput

Throughput measures the efficiency of the computer itself. Throughput is the time from the input of a request to the central processor until the output is delivered to the system

Capacity planning

- Monitors current activity and performance levels
- Anticipates future activity
- Forecasts the resources needed to provide desired levels of service

In capacity planning you can use a technique called *what-if* analysis, where you vary one or more elements in a model to see the effect on other elements

CASE Tools for System Maintenance

A CASE toolkit provides valuable tools for system evaluation and maintenance, such as

- A performance monitor
- A program analyser
- An interactive debugging analyser
- A restructuring or reengineering tool
- Automated documentation tools
- Network activity monitors
- Workload forecasting software

System Obsolescence

A system becomes obsolete when its functions are no longer required by users or when the platform becomes outmoded

Typical signs of obsolescence

- Adaptive and corrective maintenance is increasing steadily
- Operational costs or execution times are increasing rapidly, and routine perfective maintenance does not reverse the trend
- A software package is available that provides the same or additional services faster, better, and less expensively
- New technology offers a way to perform the same or additional functions more efficiently
- Maintenance changes or additions are difficult and expensive to perform
- Users request significant new features to support business requirements

Maintenance Activities

The overall cost of a system includes the systems operation and support phase. Costs include fixed operational costs and maintenance activities.

Surveys in the 80's reveals up to 60% of organisational software effort spent in maintenance

Operational costs are relatively constant, while maintenance costs vary over time

- High costs when system is implemented
- Relatively low costs during system's useful life
- High costs near end of system's useful life

Operational costs

- Supplies
- Equipment rentals
- Software leases

Maintenance activities

- Changing programs, procedures, or documentation to ensure correct performance
- Adapting the system to changing requirements
- Making the system operate more efficiently

Maintenance Activities and Roles

Maintenance activities are similar to those of development-analysis->design->coding->test

The process of maintaining IS

- Obtaining maintenance requests
- Transforming requests into changes
- Designing changes
- Implementing changes
- Deliverables and outcomes

Obtaining maintenance requests
A formal process must be established whereby uses can submit change requests. Specify an individual within the organization to collect these requests and manage their dispersal to maintenance personnel

Transforming requests into changes
Once the request is received, analysis must be conducted to gain an understanding of the scope of the request. Determine how the request will affect the current system and the duration of the project

Designing changes
A change request must be transformed into a formal design change - A Blueprint

Implementing changes
The formal design change is fed into the maintenance implementation phase

Deliverables and outcomes
The maintenance phase is basically a smaller SDLC. The deliverable is the development of a new version of the system. All prior versions of the system must be kept to enable recreating the older system if needed

People who perform maintenance have similar roles: analyst, programmers, designers
Maintenance focuses on four major aspects

- Maintenance control over the system's day-to-day function
- Maintenance control over system modifications
- Perfecting existing acceptable functions
- Preventing system performance from degrading to unacceptable levels

Three types of maintenance

- Corrective maintenance
  To fix errors
- Adaptive maintenance
  To add new capability and enhancements
- Perfective maintenance
  To improve efficiency

Corrective maintenance
Impractical to exhaustively test a large system, therefore reasonable to assume that any large system will have errors. Testing should be thorough for common cases, so errors likely to be obscure. The process of receiving reports of such errors, diagnosing the problem, and fixing it is called "corrective maintenance". Investigation, analysis, design, and testing are necessary before a solution is implemented. Typically, a user submits a systems request form with supporting evidence, if necessary. Response depends on the priority of the request . All maintenance is logged. It is used to control the day-to-day system function. The maintenance team respond to problem

- Find the failure's cause
- Make correction
- Change to requirement, design, code, test, document

Often, the initial repair is temporary: something to keep the system running, but not the best fix. Long-range change may be implemented later

Adaptive maintenance

Adds enhancements to the system. An enhancement is a new feature or capability. Adaptive maintenance often is required in a dynamic business environment. An adaptive maintenance project is like a mini-SDLC, with similar phases and tasks. Can be more difficult than new systems development, because of the constraints of an existing system. Sometimes a change introduced in one part of the system requires change to other parts, more functions are add to the system

Perfective maintenance

Involves changing an operational system to make it more efficient, reliable, or maintainable. Requests for corrective and adaptive maintenance typically come from users, while requests for perfective maintenance typically come from the IS department. As we maintain a system, we examine documents, design, code and tests, looking for opportunities for improvement. For example, as functions are added in a system the design may become confused and difficult to follow. A redesign may enhance future maintenance and make it easier for use to add new functions in the future. Involve,

- Making changes to improve some aspect for the system
- Documentation changes to clarify items
- Test suite changes to improve test coverage
- Code and design modifications to enhance readability

Preventive Maintenance

Is similar to perfective maintenance. Involve changing some aspect of the system to prevent failure. May include,

- The addition of type checking
- The enhancement of fault handling
- To make sure the system can handle all possibility
- To find potential fault that has not yet become a failure and takes and action to correct the fault before damage is done

Structured vs unstructured maintenance

Unstructured maintenance wades straight into the source code and makes changes based on that alone. Structured maintenance examines and modifies the original design, and then reworks the code to match it. Clearly structured maintenance is a more reliable and (usually) a more efficient process. Unfortunately, it's not always possible.

Managing Systems Operation and Support

Systems operation requires effective management techniques

- Maintenance team
- Configuration management (change control)
- Maintenance releases

Maintenance team

Consists of systems analysts and programmers. Systems analysts on maintenance work need

- Solid background in information technology
- Strong analytical abilities
- Solid understanding of business operations
- Effective interpersonal and communication skills

The maintenance team activities

- Understanding the system
- Locating information in the system documentation
- Keeping system documentation up to date
- Extending existing function to accommodate new or changing requirement
- Adding new functions to the system
- Finding source of system failures or problems
- Locating and correcting faults
- Answering questions about the new way the system works
- Restructuring design and code components
- Rewriting design and code components
- Deleting design and code components that are no longer useful
- Managing changes to the system as they are mode

Systems Analyst Responsibilities

- Minimize the number of latent defects - The number of latent defects refers to the number of unknown errors existing in the system after it is installed. In a system influences most of the costs associated with maintaining a system
- Take into account the number of customers
- Make sure the documentation quality is high
- Pay attention to the quality of the maintenance personnel

Managing maintenance personnel

Personnel may have limited job mobility and lack access to adequate human and technical resources. Personnel have a vested interest in effectively maintaining the system and have a better understanding of functional requirements. Documentation and testing thoroughness may suffer due to a lack of formal transfer of responsibility.

Maintenance group knows or has access to all assumptions and decisions behind the system's original design. All things cannot be documented, so the maintenance group may not know critical info about the system. Formal transfer of systems between groups improves the system and documentation quality

Measuring maintenance effectiveness

- Number of failures
- Time between each failure
- Type of failure (a more revealing method)
  Over time, logging the types of failures will provide a very clear picture of where, when, and how failures occur. Tracking the types of failures also provides important mgmt info for future projects. To effectively manage and to continuously improve, you must measure and assess performance over time.
- Controlling maintenance requests

There are various types of maintenance requests. Some correct minor or severe defects in the system. Others improve or extend system functionality. A key issue is deciding which requests to perform and which to ignore

Configuration management
Process for controlling changes in system requirements
Usually involves three steps

1. The maintenance request
2. Initial action on the request
3. Final disposition of the request

Objectives of configuration management

- Manage different versions of the system
- Organize and handle documentation

Maintenance releases
With a maintenance release methodology, all noncritical changes are held until they can be implemented at one time. Each new system version is called a release

Numbering systems

- Whole number = significant change
- After decimal = relatively minor changes or fixes

Typical problems with maintenance

- Inadequate documentation of software evolution.
- Inadequate documentation of software design and structure
- Loss of "cultural" knowledge of software due to staff turnover
- Lack of allowance for change in original software design
- Maintenance is unglamorous and may be viewed as a "punishment task"

Maintenance side-effects
Any error or undesirable behaviour that occurs as a result of modifications to a system.

- Coding side-effects (inadvertent removal of vital code, changes in semantics of code, unexpected changes in execution path)
- Data side-effects (changes in data structures render older data invalid or incomplete, changes in global constants, changes in data ranges)
- Documentation side-effects (forgetting to document code or data structure changes, changes not reflected in user manuals or interface)

Reverse engineering
Build a specification and design out of existing code
Not just for industrial theft!
Legacy systems (what if the supplier no longer exists?)
May also be required if original system documentation was lost, poor, or non-existent

Re-engineering
Preventative maintenance (prevent foreseen maintenance nightmares)
An extension of reverse engineering

Use (or modify) the reconstructed design to generate new and better source code

Software Maintenance

Often referred to as Evolution. In the 1970s maintenance accounted for around 40% of software budget. In the 1980s this figure was increased to 60%. Now the budget for maintenance is considered to be around 80% (Pressman).

Sommerville states that maintenance is roughly 1 to 4 times the development cost. Costs include repairs, portability issues, additional functionality, security etc.

Definitions of Maintenance

- Changes that have to be made to programs after delivery.
- The performance of those activities required to keep a system operational and responsive after it is accepted and placed into production
- Covers the life of a software system from the time it is installed to the time it is phased out
- Modification of a software product after delivery to correct faults to improve performance or other attributes
- Modification to code and associated documentation due to a problem or the need for improvement.

Lehman's Laws

Law of Continuing Change

A program that is used in a real-world environment necessarily must change or it becomes less useful.

Law of Increasing Complexity

As a program changes, its structure becomes more complex. Extra resources must be devoted to it.

Law of Large Program Evolution

System attributes such as size, errors are approximately invariant for each system.

Difficulties

It is often difficult to

- Trace changes in code through various versions of software
- Read someone else's code
- Understand a system without documentation
- Change code without a modification strategy
- Maintain code where design documentation or requirements are missing
- Get people to do it properly!

Regression testing

After any code changes the (partial) system should be retested with test cases that were used originally as well as any new ones. Old tests show whether changes have been made and that separate functionality is not affected. New tests are needed to demonstrate that changes have been made.

How to regression test

Look at functionality or requirements to test cases matrix and determine those modules affected by change. Re-run all the appropriate test cases. This requires traceability of requirements to tests. It also may require a design module to test case matrix. Further, take the changed modules and determine from the module coverage to test cases matrix, which tests to re-run. Further, look for Information Flow effects, aliasing, cross-referencing etc. That is, look at the data flow level and not just at the control level.